

# Samples - java (continuation)

<https://g.co/gemini/share/5d9ec78f95ed>



Created with 2.5 Pro (preview) May 16, 2025 at 07:30 PM • Published May 21, 2025 at 11:00 PM

You are an expert coding assistant specialized in software refactoring, with many years of experience analyzing code transformations.

You will be given the diffs of a commit:

```
- **Diffs:**  
diff --git  
src/main/java/com/google/maps/IStaticMapsRequest.java  
src/main/java/com/google/maps/IStaticMapsRequest.java  
new file mode 100644  
index 0000000..a44dbe0  
--- /dev/null  
+++ src/main/java/com/google/maps/IStaticMapsRequest.java  
@@ -0,0 +1,133 @@  
+package com.google.maps;  
+  
+import com.google.maps.StaticMapsRequest.ImageFormat;  
+import com.google.maps.StaticMapsRequest.Markers;  
+import com.google.maps.StaticMapsRequest.Path;  
+import  
com.google.maps.StaticMapsRequest.StaticMapType;  
+import com.google.maps.model.EncodedPolyline;  
+import com.google.maps.model.LatLng;  
+import com.google.maps.model.Size;
```

```
+  
+public interface IStaticMapsRequest {  
+  
+    /**  
+     * <code>center</code> (required if markers not  
+     * present) defines the center of the map,  
+     * equidistant from all edges of the map.  
+     *  
+     * @param location The location of the center of the  
+     * map.  
+     * @return Returns this {@code StaticMapsRequest} for  
+     * call chaining.  
+    */  
+    IStaticMapsRequest center(LatLng location);  
+  
+    /**  
+     * <code>center</code> (required if markers not  
+     * present) defines the center of the map,  
+     * equidistant from all edges of the map.  
+     *  
+     * @param location The location of the center of the  
+     * map.  
+     * @return Returns this {@code StaticMapsRequest} for  
+     * call chaining.  
+    */  
+    IStaticMapsRequest center(String location);  
+  
+    /**  
+     * <code>zoom</code> (required if markers not present)  
+     * defines the zoom level of the map, which  
+     * determines the magnification level of the map.  
+     *  
+     * @param zoom The zoom level of the region.  
+     * @return Returns this {@code StaticMapsRequest} for  
+     * call chaining.  
+    */
```

```
+     IStaticMapsRequest zoom(int zoom);  
+  
+     /**  
+      * <code>size</code> defines the rectangular  
dimensions of the map image.  
+      *  
+      * @param size The Size of the static map.  
+      * @return Returns this {@code StaticMapsRequest} for  
call chaining.  
+     */  
+     IStaticMapsRequest size(Size size);  
+  
+     /**  
+      * <code>scale</code> affects the number of pixels that  
are returned. Setting <code>scale</code>  
+      * to 2 returns twice as many pixels as  
<code>scale</code> set to 1 while retaining the same  
+      * coverage area and level of detail (i.e. the contents of  
the map doesn't change).  
+      *  
+      * @param scale The scale of the static map.  
+      * @return Returns this {@code StaticMapsRequest} for  
call chaining.  
+     */  
+     IStaticMapsRequest scale(int scale);  
+  
+     /**  
+      * <code>format</code> defines the format of the  
resulting image. By default, the Google Static  
+      * Maps API creates PNG images. There are several  
possible formats including GIF, JPEG and PNG  
+      * types.  
+      *  
+      * @param format The format of the static map.  
+      * @return Returns this {@code StaticMapsRequest} for  
call chaining.
```

```
+      */
+      IStaticMapsRequest format(ImageFormat format);
+
+
+      /**
+       * <code>maptypes</code> defines the type of map to
construct.
+
+      *
+      * @param maptype The map type of the static map.
+      * @return Returns this {@code StaticMapsRequest} for
call chaining.
+
+      */
+      IStaticMapsRequest maptype(StaticMapType maptype);
+
+
+      /**
+       * <code>region</code> defines the appropriate
borders to display, based on geo-politi
cal
+
+       * sensitivities. Accepts a region code specified as a
two-character ccTLD ('top-level domain')
+
+       * value.
+
+       *
+       * @param region The region of the static map.
+       * @return Returns this {@code StaticMapsRequest} for
call chaining.
+
+      */
+      IStaticMapsRequest region(String region);
+
+
+      /**
+       * <code>markers</code> parameter defines a set of
one or more markers (map pins) at a
set of
+
+       * locations. Each marker defined within a single
markers declaration must exhibit the same visual
+
+       * style; if you wish to display markers with different
styles, you will need to supply multiple
+
+       * markers parameters with separate style information.
```

```
+      *
+      * @param markers A group of markers with the same
style.
+      * @return Returns this {@code StaticMapsRequest} for
call chaining.
+
+      */
+
+      IStaticMapsRequest markers(Markers markers);
+
+
+      /**
+      * The <code>path</code> parameter defines a set of
one or more locations connected by a path to
+      * overlay on the map image.
+
+      *
+      * @param path A path to render atop the map.
+      * @return Returns this {@code StaticMapsRequest} for
call chaining.
+
+      */
+
+      IStaticMapsRequest path(Path path);
+
+
+      /**
+      * The <code>path</code> parameter defines a set of
one or more locations connected by a path to
+      * overlay on the map image. This variant of the method
accepts the path as an EncodedPolyline.
+
+      *
+      * @param path A path to render atop the map, as an
EncodedPolyline.
+      * @return Returns this {@code StaticMapsRequest} for
call chaining.
+
+      */
+
+      IStaticMapsRequest path(EncodedPolyline path);
+
+
+      /**
+      * <code>visible</code> instructs the Google Static
Maps API service to construct a map such that
+      * the existing locations remain visible.
```

```

+      *
+      * @param visibleLocation The location to be made
visible in the requested Static Map.+      * @return Returns
this {@code StaticMapsRequest} for call chaining.
+      */
+      IStaticMapsRequest visible(LatLng visibleLocation);
+
+      /**
+       * <code>visible</code> instructs the Google Static
Maps API service to construct a map such that
+       * the existing locations remain visible.
+       *
+       * @param visibleLocation The location to be made
visible in the requested Static Map.+      * @return Returns
this {@code StaticMapsRequest} for call chaining.
+      */
+      IStaticMapsRequest visible(String visibleLocation);
+
+}
\ No newline at end of file
diff --git
src/main/java/com/google/maps/StaticMapsRequest.java
src/main/java/com/google/maps/StaticMapsRequest.java
index 327a8478..eddeac45 100644
--- src/main/java/com/google/maps/StaticMapsRequest.java
+++ src/main/java/com/google/maps/StaticMapsRequest.java
@@ -26,7 +26,7 @@ import java.util.List;
import java.util.regex.Pattern;

public class StaticMapsRequest
- extends PendingResultBase<ImageResult,
StaticMapsRequest, ImageResult.Response> {
+ extends PendingResultBase<ImageResult,
StaticMapsRequest, ImageResult.Response> implements
IStaticMapsRequest {

```

```
static final ApiConfig API_CONFIG = new
ApiConfig("/maps/api/staticmap");

@@ -54,7 +54,7 @@ public class StaticMapsRequest
 * @param location The location of the center of the map.
 * @return Returns this {@code StaticMapsRequest} for call
chaining.
*/
- public StaticMapsRequest center(LatLng location) {
+ public IStaticMapsRequest center(LatLng location) {
    return param("center", location);
}

@@ -65,7 +65,7 @@ public class StaticMapsRequest
 * @param location The location of the center of the map.
 * @return Returns this {@code StaticMapsRequest} for call
chaining.
*/
- public StaticMapsRequest center(String location) {
+ public IStaticMapsRequest center(String location) {
    return param("center", location);
}

@@ -76,7 +76,7 @@ public class StaticMapsRequest
 * @param zoom The zoom level of the region.
 * @return Returns this {@code StaticMapsRequest} for call
chaining.
*/
- public StaticMapsRequest zoom(int zoom) {
+ public IStaticMapsRequest zoom(int zoom) {
    return param("zoom", zoom);
}

@@ -98,7 +98,7 @@ public class StaticMapsRequest
 * @param scale The scale of the static map.
```

```
* @return Returns this {@code StaticMapsRequest} for call  
chaining.
```

```
*/
```

```
- public StaticMapsRequest scale(int scale) {
```

```
+ public IStaticMapsRequest scale(int scale) {
```

```
    return param("scale", scale);
```

```
}
```

```
@@ -130,7 +130,7 @@ public class StaticMapsRequest
```

```
* @param format The format of the static map.
```

```
* @return Returns this {@code StaticMapsRequest} for call  
chaining.
```

```
*/
```

```
- public StaticMapsRequest format(ImageFormat format) {
```

```
+ public IStaticMapsRequest format(ImageFormat format) {
```

```
    return param("format", format);
```

```
}
```

```
@@ -152,7 +152,7 @@ public class StaticMapsRequest
```

```
* @param maptype The map type of the static map.
```

```
* @return Returns this {@code StaticMapsRequest} for call  
chaining.
```

```
*/
```

```
- public StaticMapsRequest maptype(StaticMapType  
maptype) {
```

```
+ public IStaticMapsRequest maptype(StaticMapType  
maptype) {
```

```
    return param("maptype", maptype);
```

```
}
```

```
@@ -164,7 +164,7 @@ public class StaticMapsRequest
```

```
* @param region The region of the static map.
```

```
* @return Returns this {@code StaticMapsRequest} for call  
chaining.
```

```
*/
```

```
- public StaticMapsRequest region(String region) {
```

```
+ public IStaticMapsRequest region(String region) {  
    return param("region", region);  
}  
  
@@ -328,7 +328,7 @@ public class StaticMapsRequest  
    * @param markers A group of markers with the same style.  
    * @return Returns this {@code StaticMapsRequest} for call  
chaining.  
*/  
- public StaticMapsRequest markers(Markers markers) {  
+ public IStaticMapsRequest markers(Markers markers) {  
    return paramAddToList("markers", markers);  
}  
  
@@ -431,7 +431,7 @@ public class StaticMapsRequest  
    * @param path A path to render atop the map.  
    * @return Returns this {@code StaticMapsRequest} for call  
chaining.  
*/  
- public StaticMapsRequest path(Path path) {  
+ public IStaticMapsRequest path(Path path) {  
    return paramAddToList("path", path);  
}  
  
@@ -442,7 +442,7 @@ public class StaticMapsRequest  
    * @param path A path to render atop the map, as an  
EncodedPolyline.  
    * @return Returns this {@code StaticMapsRequest} for call  
chaining.  
*/  
- public StaticMapsRequest path(EncodedPolyline path) {  
+ public IStaticMapsRequest path(EncodedPolyline path) {  
    return paramAddToList("path", "enc:" +  
path.getEncodedPath());  
}
```

```
@@ -453,7 +453,7 @@ public class StaticMapsRequest
    * @param visibleLocation The location to be made visible in
    the requested Static Map.
    * @return Returns this {@code StaticMapsRequest} for call
    chaining.
    */
- public StaticMapsRequest visible(LatLng visibleLocation) {
+ public IStaticMapsRequest visible(LatLng visibleLocation) {
    return param("visible", visibleLocation);
}
```

```
@@ -464,7 +464,7 @@ public class StaticMapsRequest
    * @param visibleLocation The location to be made visible in
    the requested Static Map.
    * @return Returns this {@code StaticMapsRequest} for call
    chaining.
    */
- public StaticMapsRequest visible(String visibleLocation) {
+ public IStaticMapsRequest visible(String visibleLocation) {
    return param("visible", visibleLocation);
}
```

Your task is to identify which refactoring type(s) have been applied in transforming the original program into the new version. Use only the following list of predefined refactorings:

(Add Method Parameter) – Introduces a new parameter to an existing method.

(Remove Method Parameter) – Eliminates an existing parameter from a method signature.

(Rename Method) – Changes the name of a method while preserving its behavior.

(Rename Class) – Changes the name of a class without altering its structure.

(Rename Package) – Changes the name of a package declaration.

(Rename Field) – Changes the name of a class or instance variable.

(Extract Class) – Moves a group of related fields and methods from an existing class into a newly created class.

(Extract Superclass) – Creates a new superclass to encapsulate shared attributes and behavior from two or more existing classes.

(Inline Method) – Replaces a method call with the method's body.

(Pull Up Method) – Pull a method from a child class (subclass) in the original code to its parent class (superclass) in the transformed version.

(Push Down Method) – Push a method from a parent class (superclass) in the original code to one or more of its child classes (subclasses) in the transformed version.

(Pull Up Field) – Pull a field from a child class (subclass) in the original code to its parent class (superclass) in the transformed version.

(Push Down Field) – Push a field from a parent class (superclass) in the original code to one or more of its child classes (subclasses) in the transformed version.

(Inline Class) – Merges a class into another when it is too small or redundant.

(Extract Interface) – Creates a new interface from existing method(s) in a class.

(Move Method) – Relocates a method to a more appropriate class.

(Move Field) – Relocates a field to a more appropriate class.

(Replace Magic Number with Constant) – Replaces a literal number with a named constant.

(Encapsulate Field) – Change a field access from not private to private and create get and set methods.

**\*\*Instructions:\*\***

1. Begin your response with a bullet-point list of the refactoring type(s) applied.
2. Then, briefly justify each identified refactoring with reference to the specific code changes.
3. Only include refactorings from the list above.
4. Be concise but precise in your explanations.

Do not generate explanations unrelated to the given transformation.

⋮

You are an expert coding assistant specialized in software refactoring, with many years of experience analyzing code transformations.

You will be given the diffs of a commit:

```
- **Diffs:**  
diff --git spring-boot-project/spring-boot-  
actuator/src/test/java/org/springframework/boot/actuate/metri  
cs/web/servlet/ILongTaskTimingHandlerInterceptorTests.java  
spring-boot-project/spring-boot-  
actuator/src/test/java/org/springframework/boot/actuate/metri  
cs/web/servlet/ILongTaskTimingHandlerInterceptorTests.java  
new file mode 100644  
index 0000000000..96644cf651  
--- /dev/null  
+++ spring-boot-project/spring-boot-  
actuator/src/test/java/org/springframework/boot/actuate/metri  
cs/web/servlet/ILongTaskTimingHandlerInterceptorTests.java  
@@ -0,0 +1,5 @@
```

```
+package  
org.springframework.boot.actuate.metrics.web.servlet;  
+  
+interface ILongTaskTimingHandlerInterceptorTests {  
+  
+}  
\ No newline at end of file  
diff --git spring-boot-project/spring-boot-  
actuator/src/test/java/org/springframework/boot/actuate/metri  
cs/web/servlet/LongTaskTimingHandlerInterceptorTests.java  
spring-boot-project/spring-boot-  
actuator/src/test/java/org/springframework/boot/actuate/metri  
cs/web/servlet/LongTaskTimingHandlerInterceptorTests.java  
index 1279f3131a..a5a840edd4 100644  
--- spring-boot-project/spring-boot-  
actuator/src/test/java/org/springframework/boot/actuate/metri  
cs/web/servlet/LongTaskTimingHandlerInterceptorTests.java  
+++ spring-boot-project/spring-boot-  
actuator/src/test/java/org/springframework/boot/actuate/metri  
cs/web/servlet/LongTaskTimingHandlerInterceptorTests.java  
@@ -65,7 +65,7 @@ import static  
org.springframework.test.web.servlet.result.MockMvcResultM  
atchers.  
*/  
@ExtendWith(SpringExtension.class)  
@WebAppConfiguration  
-class LongTaskTimingHandlerInterceptorTests {  
+class LongTaskTimingHandlerInterceptorTests implements  
ILongTaskTimingHandlerInterceptorTests {  
  
    @Autowired  
    private SimpleMeterRegistry registry;
```

Your task is to identify which refactoring type(s) have been applied in transforming the original program into the new version. Use only the following list of predefined refactorings:

- (Add Method Parameter) – Introduces a new parameter to an existing method.
- (Remove Method Parameter) – Eliminates an existing parameter from a method signature.
- (Rename Method) – Changes the name of a method while preserving its behavior.
- (Rename Class) – Changes the name of a class without altering its structure.
- (Rename Package) – Changes the name of a package declaration.
- (Rename Field) – Changes the name of a class or instance variable.
- (Extract Class) – Moves a group of related fields and methods from an existing class into a newly created class.
- (Extract Superclass) – Creates a new superclass to encapsulate shared attributes and behavior from two or more existing classes.
- (Inline Method) – Replaces a method call with the method's body.
- (Pull Up Method) – Pull a method from a child class (subclass) in the original code to its parent class (superclass) in the transformed version.
- (Push Down Method) – Push a method from a parent class (superclass) in the original code to one or more of its child classes (subclasses) in the transformed version.
- (Pull Up Field) – Pull a field from a child class (subclass) in the original code to its parent class (superclass) in the transformed version.
- (Push Down Field) – Push a field from a parent class (superclass) in the original code to one or more of its child classes (subclasses) in the transformed version.
- (Inline Class) – Merges a class into another when it is too small or redundant.
- (Extract Interface) – Creates a new interface from existing method(s) in a class.

(Move Method) – Relocates a method to a more appropriate class.

(Move Field) – Relocates a field to a more appropriate class.

(Replace Magic Number with Constant) – Replaces a literal number with a named constant.

(Encapsulate Field) – Change a field access from not private to private and create get and set methods.

**\*\*Instructions:\*\***

1. Begin your response with a bullet-point list of the refactoring type(s) applied.

2. Then, briefly justify each identified refactoring with reference to the specific code changes.

3. Only include refactorings from the list above.

4. Be concise but precise in your explanations.

Do not generate explanations unrelated to the given transformation.

⋮

You are an expert coding assistant specialized in software refactoring, with many years of experience analyzing code transformations.

You will be given the diffs of a commit:

- **\*\*Diffs:\*\***

diff --git

src/main/java/com/google/maps/FindPlaceFromTextRequest.java  
va

src/main/java/com/google/maps/FindPlaceFromTextRequest.java  
va

index 667738b1..4aa08d84 100644

```
---  
src/main/java/com/google/maps/FindPlaceFromTextRequest.java  
va  
+++  
src/main/java/com/google/maps/FindPlaceFromTextRequest.java  
va  
@@ -85,17 +85,6 @@ public class FindPlaceFromTextRequest  
    return param("fields", StringJoin.join(',', fields));  
}  
  
- /**  
- * Prefer results in a specified area, by specifying either a  
radius plus lat/lng, or two lat/lng  
- * pairs representing the points of a rectangle.  
- *  
- * @param locationBias The location bias for this request.  
- * @return Returns {@code FindPlaceFromTextRequest} for  
call chaining.  
- */  
- public FindPlaceFromTextRequest  
locationBias(LocationBias locationBias) {  
-    return param("locationbias", locationBias);  
- }  
-  
-  
@Override  
protected void validateRequest() {  
    if (!params().containsKey("input")) {  
diff --git src/test/java/com/google/maps/PlacesApiTest.java  
src/test/java/com/google/maps/PlacesApiTest.java  
index cdba990c..4e912da6 100644  
--- src/test/java/com/google/maps/PlacesApiTest.java  
+++ src/test/java/com/google/maps/PlacesApiTest.java  
@@ -22,6 +22,7 @@ import static  
org.junit.Assert.assertNotNull;  
import static org.junit.Assert.assertTrue;
```

```
import
com.google.maps.FindPlaceFromTextRequest.InputType;
+import
com.google.maps.FindPlaceFromTextRequest.LocationBias;
import
com.google.maps.FindPlaceFromTextRequest.LocationBiasCircular;
import
com.google.maps.FindPlaceFromTextRequest.LocationBiasIP;
import
com.google.maps.FindPlaceFromTextRequest.LocationBiasPoint;
@@ -918,18 +919,18 @@ public class PlacesApiTest {
    new
LocalTestServerContext(findPlaceFromTextMuseumOfContem
poraryArt)) {

    String input = "Museum of Contemporary Art Australia";
+    LocationBias locationBias = new LocationBiasIP();

    FindPlaceFromText response =
        PlacesApi.findPlaceFromText(sc.context, input,
InputType.TEXT_QUERY)
    -
        .fields(
    -
        FindPlaceFromTextRequest.FieldMask.BUSINESS_STATUS,
    -
        FindPlaceFromTextRequest.FieldMask.PHOTOS,
    -
        FindPlaceFromTextRequest.FieldMask.FORMATTED_ADDRESS
    ,
    -
        FindPlaceFromTextRequest.FieldMask.NAME,
    -
        FindPlaceFromTextRequest.FieldMask.RATING,
    -
        FindPlaceFromTextRequest.FieldMask.OPENING_HOURS,
    -
        FindPlaceFromTextRequest.FieldMask.GEOMETRY)
    -
        .locationBias(new LocationBiasIP())
```

```
+           .fields(
+
+           FindPlaceFromTextRequest.FieldMask.BUSINESS_STATUS,
+           FindPlaceFromTextRequest.FieldMask.PHOTOS,
+
+           FindPlaceFromTextRequest.FieldMask.FORMATTED_ADDRESS,
+
+           FindPlaceFromTextRequest.FieldMask.NAME,
+           FindPlaceFromTextRequest.FieldMask.RATING,
+
+           FindPlaceFromTextRequest.FieldMask.OPENING_HOURS,
+
+           FindPlaceFromTextRequest.FieldMask.GEOMETRY).param("locationbias", locationBias)
+
.await();

sc.assertParamValue(input, "input");
@@ -963,16 +964,16 @@ public class PlacesApiTest {
    new
LocalTestServerContext(findPlaceFromTextMuseumOfContemporaryArt)) {

    String input = "Museum of Contemporary Art Australia";
+    LocationBias locationBias = new LocationBiasPoint(new
LatLang(1, 2));

    PlacesApi.findPlaceFromText(sc.context, input,
InputType.TEXT_QUERY)
-
    .fields(
-
        FindPlaceFromTextRequest.FieldMask.PHOTOS,
-
        FindPlaceFromTextRequest.FieldMask.FORMATTED_ADDRESS,
-
        FindPlaceFromTextRequest.FieldMask.NAME,
-
        FindPlaceFromTextRequest.FieldMask.RATING,
```

```
-      FindPlaceFromTextRequest.FieldMask.OPENING_HOURS,
-      FindPlaceFromTextRequest.FieldMask.GEOMETRY)
-      .locationBias(new LocationBiasPoint(new LatLng(1, 2)))
+      .fields(
+          FindPlaceFromTextRequest.FieldMask.PHOTOS,
+
+          FindPlaceFromTextRequest.FieldMask.FORMATTED_ADDRESS,
+
+          FindPlaceFromTextRequest.FieldMask.NAME,
+
+          FindPlaceFromTextRequest.FieldMask.RATING,
+
+          FindPlaceFromTextRequest.FieldMask.OPENING_HOURS,
+
+          FindPlaceFromTextRequest.FieldMask.GEOMETRY).param("locationbias", locationBias)
+
+          .await();

sc.assertParamValue(input, "input");
@@ -988,16 +989,16 @@ public class PlacesApiTest {
    new
LocalTestServerContext(findPlaceFromTextMuseumOfContemporaryArt)) {

    String input = "Museum of Contemporary Art Australia";
+    LocationBias locationBias = new
LocationBiasCircular(new LatLng(1, 2), 3000);

    PlacesApi.findPlaceFromText(sc.context, input,
InputType.TEXT_QUERY)
-
-        .fields(
-            FindPlaceFromTextRequest.FieldMask.PHOTOS,
-
-
FindPlaceFromTextRequest.FieldMask.FORMATTED_ADDRESS
,
-
-            FindPlaceFromTextRequest.FieldMask.NAME,
```

```
-      FindPlaceFromTextRequest.FieldMask.RATING,
-
FindPlaceFromTextRequest.FieldMask.OPENING_HOURS,
-      FindPlaceFromTextRequest.FieldMask.GEOMETRY)
-      .locationBias(new LocationBiasCircular(new LatLng(1,
2), 3000))
+      .fields(
+      FindPlaceFromTextRequest.FieldMask.PHOTOS,
+
FindPlaceFromTextRequest.FieldMask.FORMATTED_ADDRES
S,
+      FindPlaceFromTextRequest.FieldMask.NAME,
+      FindPlaceFromTextRequest.FieldMask.RATING,
+
FindPlaceFromTextRequest.FieldMask.OPENING_HOURS,
+
FindPlaceFromTextRequest.FieldMask.GEOMETRY).param("lo
cationbias", locationBias)
.await();

sc.assertParamValue(input, "input");
@@ -1013,16 +1014,16 @@ public class PlacesApiTest {
    new
LocalTestServerContext(findPlaceFromTextMuseumOfContem
poraryArt)) {

    String input = "Museum of Contemporary Art Australia";
+      LocationBias locationBias = new
LocationBiasRectangular(new LatLng(1, 2), new LatLng(3,
4));

    PlacesApi.findPlaceFromText(sc.context, input,
InputType.TEXT_QUERY)
-      .fields(
-      FindPlaceFromTextRequest.FieldMask.PHOTOS,
```

```
-      FindPlaceFromTextRequest.FieldMask.FORMATTED_ADDRESS
-
-      FindPlaceFromTextRequest.FieldMask.NAME,
-      FindPlaceFromTextRequest.FieldMask.RATING,
-
-      FindPlaceFromTextRequest.FieldMask.OPENING_HOURS,
-      FindPlaceFromTextRequest.FieldMask.GEOMETRY)
-      .locationBias(new LocationBiasRectangular(new
LatLng(1, 2), new LatLng(3, 4)))
+      .fields(
+      FindPlaceFromTextRequest.FieldMask.PHOTOS,
+
+      FindPlaceFromTextRequest.FieldMask.FORMATTED_ADDRES
S,
+      FindPlaceFromTextRequest.FieldMask.NAME,
+      FindPlaceFromTextRequest.FieldMask.RATING,
+
+      FindPlaceFromTextRequest.FieldMask.OPENING_HOURS,
+
FindPlaceFromTextRequest.FieldMask.GEOMETRY).param("lo
cationbias", locationBias)
.await();

sc.assertParamValue(input, "input");
```

Your task is to identify which refactoring type(s) have been applied in transforming the original program into the new version. Use only the following list of predefined refactorings:

(Add Method Parameter) – Introduces a new parameter to an existing method.

(Remove Method Parameter) – Eliminates an existing parameter from a method signature.

(Rename Method) – Changes the name of a method while preserving its behavior.

(Rename Class) – Changes the name of a class without altering its structure.

(Rename Package) – Changes the name of a package declaration.

(Rename Field) – Changes the name of a class or instance variable.

(Extract Class) – Moves a group of related fields and methods from an existing class into a newly created class.

(Extract Superclass) – Creates a new superclass to encapsulate shared attributes and behavior from two or more existing classes.

(Inline Method) – Replaces a method call with the method's body.

(Pull Up Method) – Pull a method from a child class (subclass) in the original code to its parent class (superclass) in the transformed version.

(Push Down Method) – Push a method from a parent class (superclass) in the original code to one or more of its child classes (subclasses) in the transformed version.

(Pull Up Field) – Pull a field from a child class (subclass) in the original code to its parent class (superclass) in the transformed version.

(Push Down Field) – Push a field from a parent class (superclass) in the original code to one or more of its child classes (subclasses) in the transformed version.

(Inline Class) – Merges a class into another when it is too small or redundant.

(Extract Interface) – Creates a new interface from existing method(s) in a class.

(Move Method) – Relocates a method to a more appropriate class.

(Move Field) – Relocates a field to a more appropriate class.

(Replace Magic Number with Constant) – Replaces a literal number with a named constant.

(Encapsulate Field) – Change a field access from not private to private and create get and set methods.

**\*\*Instructions:\*\***

1. Begin your response with a bullet-point list of the refactoring type(s) applied.
2. Then, briefly justify each identified refactorng with reference to the specific code changes.
3. Only include refactorings from the list above.
4. Be concise but precise in your explanations.

Do not generate explanations unrelated to the given transformation.

⋮

```
diff --git gobblin-
runtime/src/main/java/org/apache/gobblin/runtime/JobLauncherFactory.java gobblin-
runtime/src/main/java/org/apache/gobblin/runtime/JobLauncherFactory.java
index ec0a7e487..a67e9a155 100644
--- gobblin-
runtime/src/main/java/org/apache/gobblin/runtime/JobLauncherFactory.java
+++ gobblin-
runtime/src/main/java/org/apache/gobblin/runtime/JobLauncherFactory.java
@@ -107,21 +107,6 @@ public class JobLauncherFactory {
    return newJobLauncher(sysProps, jobProps,
launcherTypeValue, instanceBroker, metadataTags); }
```

- /\*\*
- \* Creates a new instance for a JobLauncher with a given
type

- \* @param sysProps the system/environment properties
- \* @param jobProps the job properties
- \* @param launcherTypeValue the type of the launcher; either a {@link JobLauncherType} value or
  - \* the name of the class that extends {@link AbstractJobLauncher} and has a constructor
  - \* that has a single Properties parameter..
  - \* @return the JobLauncher instance
  - \* @throws RuntimeException if the instantiation fails
- \*/
- public static JobLauncher newJobLauncher(Properties sysProps, Properties jobProps, String launcherTypeValue, SharedResourcesBroker<GobblinScopeTypes> instanceBroker) {
  - return newJobLauncher(sysProps, jobProps, launcherTypeValue, instanceBroker, ImmutableList.of());
- }
- 
- ```
/**
```
- ```
* Creates a new instance for a JobLauncher with a given type
```
- ```
* @param sysProps the system/environment properties
```
- diff --git gobblin-runtime/src/main/java/org/apache/gobblin/runtime/job\_exec/JobLauncherExecutionDriver.java gobblin-runtime/src/main/java/org/apache/gobblin/runtime/job\_exec/JobLauncherExecutionDriver.java
- index ea79cc49c..111332506 100644 --- gobblin-runtime/src/main/java/org/apache/gobblin/runtime/job\_exec/JobLauncherExecutionDriver.java

```
+++ gobblin-
runtime/src/main/java/org/apache/gobblin/runtime/job_exec/JobLauncherExecutionDriver.java
@@ -35,6 +35,7 @@ import org.slf4j.LoggerFactory;
import com.google.common.annotations.VisibleForTesting;
import com.google.common.base.Optional;
import com.google.common.base.Preconditions;
+import com.google.common.collect.ImmutableList;
import com.google.common.io.Closer;
import com.google.common.util.concurrent.ExecutionList;
import com.typesafe.config.ConfigFactory;
@@ -170,8 +171,10 @@ public class
JobLauncherExecutionDriver extends
FutureTask<JobExecutionResult> {
    private static JobLauncher createLauncher(Configurable
_sysConfig, JobSpec _jobSpec, Logger _log,
    Optional<String> jobLauncherType,
SharedResourcesBroker<GobblinScopeTypes>
instanceBroker) {
    if (jobLauncherType.isPresent()) {
-        return
JobLauncherFactory.newJobLauncher(_sysConfig.getConfigAsProperties(),
-        _jobSpec.getConfigAsProperties(),
jobLauncherType.get(), instanceBroker);
+        Properties sysProps =
_sysConfig.getConfigAsProperties();
+        Properties jobProps =
_jobSpec.getConfigAsProperties();
+        String launcherTypeValue = jobLauncherType.get();
+        return JobLauncherFactory.newJobLauncher(sysProps,
jobProps, launcherTypeValue, instance
Broker, ImmutableList.of());
    }
    else {
        _log.info("Creating auto jobLauncher for " + _jobSpec);
    }
}
```

You are an expert coding assistant specialized in software refactoring, with many years of experience analyzing code transformations.

You will be given the diffs of a commit:

- **\*\*Diffs:\*\***

```
diff --git spring-boot-project/spring-boot-
actuator/src/main/java/org/springframework/boot/actuate/heal
th/Health.java spring-boot-project/spring-boot-
actuator/src/main/java/org/springframework/boot/actuate/heal
th/Health.java
index 438bec08c6..602bd65f83 100644
--- spring-boot-project/spring-boot-
actuator/src/main/java/org/springframework/boot/actuate/heal
th/Health.java
+++ spring-boot-project/spring-boot-
actuator/src/main/java/org/springframework/boot/actuate/heal
th/Health.java
@@ -23,6 +23,10 @@ import java.util.Map;
import com.fasterxml.jackson.annotation.JsonInclude;
import
com.fasterxml.jackson.annotation.JsonInclude.Include;

+import org.neo4j.driver.Result;
+import org.neo4j.driver.Session;
+import org.neo4j.driver.summary.ResultSummary;
+import
org.springframework.boot.actuate.neo4j.Neo4jHealthIndicato
r;
```

```
import org.springframework.util.Assert;

/**
@@ -329,6 +333,17 @@ public final class Health extends
HealthComponent {

    return new Health(this);

}

+
+    public void
runHealthCheckQuery(Neo4jHealthIndicator
neo4jHealthIndicator) {
+
        // We use WRITE here to make sure UP is
returned for a server that supports
+
        // all possible workloads
+
        try (Session session =
neo4jHealthIndicator.driver.session(Neo4jHealthIndicator.DEF
AULT_SE
SSION_CONFIG)) {
+
            Result result =
session.run(Neo4jHealthIndicator.CYPHER);
+
            String edition =
result.single().get("edition").asString();
+
            ResultSummary resultSummary =
result.consume();
+
            neo4jHealthIndicator.healthDetailsHandler.addHealthDetails(t
his, edition, resultSummary);
+
        }
+
    }

}

diff --git spring-boot-project/spring-boot-
actuator/src/main/java/org/springframework/boot/actuate/neo
4j/Neo4jHealthDetailsHandler.java spring-boot-
```

```
project/spring-boot-
actuator/src/main/java/org/springframework/boot/actuate/neo
4j/Neo4jHealthDetailsHandler.java
index 5bf908a3cb..c4cbdd2288 100644
--- spring-boot-project/spring-boot-
actuator/src/main/java/org/springframework/boot/actuate/neo
4j/Neo4jHealthDetailsHandler.java
+++ spring-boot-project/spring-boot-
actuator/src/main/java/org/springframework/boot/actuate/neo
4j/Neo4jHealthDetailsHandler.java
@@ -28,7 +28,7 @@ import
org.springframework.util.StringUtils;
*
* @author Stephane Nicoll
*/
-class Neo4jHealthDetailsHandler {
+public class Neo4jHealthDetailsHandler {

    /**
     * Add health details for the specified {@link
ResultSummary} and {@code edition}.
@@ -36,7 +36,7 @@ class Neo4jHealthDetailsHandler {
     * @param edition the edition of the server
     * @param resultSummary server information
     */
-    void addHealthDetails(Builder builder, String edition,
ResultSummary resultSummary) {
+    public void addHealthDetails(Builder builder, String
edition, ResultSummary resultSummary) {
        ServerInfo serverInfo = resultSummary.server();
        builder.up().withDetail("server", serverInfo.version() +
"@" + serverInfo.address()).withDetail("edition",
edition);
diff --git spring-boot-project/spring-boot-
actuator/src/main/java/org/springframework/boot/actuate/neo
4j/Neo4jHealthIndicator.java spring-boot-project/spring-boot-
```

```
actuator/src/main/java/org/springframework/boot/actuate/neo  
4j/Neo4jHealthIndicator.java  
index 782be4cb2b..85bb94c40b 100644  
--- spring-boot-project/spring-boot-  
actuator/src/main/java/org/springframework/boot/actuate/neo  
4j/Neo4jHealthIndicator.java  
+++ spring-boot-project/spring-boot-  
actuator/src/main/java/org/springframework/boot/actuate/neo  
4j/Neo4jHealthIndicator.java  
@@ -20,12 +20,8 @@ import  
org.apache.commons.logging.Log;  
import org.apache.commons.logging.LogFactory;  
import org.neo4j.driver.AccessMode;  
import org.neo4j.driver.Driver;  
-import org.neo4j.driver.Result;  
-import org.neo4j.driver.Session;  
import org.neo4j.driver.SessionConfig;  
import org.neo4j.driver.exceptions.SessionExpiredException;  
-import org.neo4j.driver.summary.ResultSummary;  
-  
import  
org.springframework.boot.actuate.health.AbstractHealthIndic  
ator;  
import org.springframework.boot.actuate.health.Health;  
import  
org.springframework.boot.actuate.health.HealthIndicator;  
@@ -46,7 +42,7 @@ public class Neo4jHealthIndicator  
extends AbstractHealthIndicator {  
/**  
 * The Cypher statement used to verify Neo4j is up.  
 */  
- static final String CYPHER = "CALL dbms.components()  
YIELD name, edition WHERE name = 'Neo4j Kernel' RETURN  
edition";  
+ public static final String CYPHER = "CALL  
dbms.components() YIELD name, edition WHERE name =
```

```

'Neo4j Kernel' RETURN edition";

/**
 * Message logged before retrying a health check.
@@ -56,12 +52,12 @@ public class Neo4jHealthIndicator
extends AbstractHealthIndicator {

/**
 * The default session config to use while connecting.
*/
- static final SessionConfig DEFAULT_SESSION_CONFIG =
SessionConfig.builder().withDefaultAccessMode(AccessMode
.WRITE)
+ public static final SessionConfig
DEFAULT_SESSION_CONFIG =
SessionConfig.builder().withDefaultAccessMode(AccessMode
.WRITE)
.build();

-
private final Driver driver;
+
public final Driver driver;

-
private final Neo4jHealthDetailsHandler
healthDetailsHandler;
+
public final Neo4jHealthDetailsHandler
healthDetailsHandler;

public Neo4jHealthIndicator(Driver driver) {
    super("Neo4j health check failed");
@@ -73,12 +69,12 @@ public class Neo4jHealthIndicator
extends AbstractHealthIndicator {

    protected void doHealthCheck(Health.Builder builder) {
        try {
            try {
-
runHealthCheckQuery(builder);
+
builder.runHealthCheckQuery(this);
}

```

```

        catch (SessionExpiredException ex) {
            // Retry one time when the session has been
            expired
            logger.warn(MESSAGE_SESSION_EXPIRED);
            -
            runHealthCheckQuery(builder);
            +
            builder.runHealthCheckQuery(this);
        }
    }

    catch (Exception ex) {
        @@ -86,15 +82,4 @@ public class Neo4jHealthIndicator
        extends AbstractHealthIndicator {
    }
}

-
private void runHealthCheckQuery(Health.Builder
builder) {
-
    // We use WRITE here to make sure UP is returned for
    a server that supports
-
    // all possible workloads
-
    try (Session session =
this.driver.session(DEFAULT_SESSION_CONFIG)) {
-
        Result result = session.run(CYPHER);
-
        String edition =
result.single().get("edition").asString();
-
        ResultSummary resultSummary =
result.consume();
-
-
        this.healthDetailsHandler.addHealthDetails(builder, edition,
resultSummary);
-
    }
}
-
}

```

Your task is to identify which refactoring type(s) have been applied in transforming the original program into the new

version. Use only the following list of predefined refactorings:

(Add Method Parameter) – Introduces a new parameter to an existing method.

(Remove Method Parameter) – Eliminates an existing parameter from a method signature.

(Rename Method) – Changes the name of a method while preserving its behavior.

(Rename Class) – Changes the name of a class without altering its structure.

(Rename Package) – Changes the name of a package declaration.

(Rename Field) – Changes the name of a class or instance variable.

(Extract Class) – Moves a group of related fields and methods from an existing class into a newly created class.

(Extract Superclass) – Creates a new superclass to encapsulate shared attributes and behavior from two or more existing classes.

(Inline Method) – Replaces a method call with the method's body.

(Pull Up Method) – Pull a method from a child class (subclass) in the original code to its parent class (superclass) in the transformed version.

(Push Down Method) – Push a method from a parent class (superclass) in the original code to one or more of its child classes (subclasses) in the transformed version.

(Pull Up Field) – Pull a field from a child class (subclass) in the original code to its parent class (superclass) in the transformed version.

(Push Down Field) – Push a field from a parent class (superclass) in the original code to one or more of its child classes (subclasses) in the transformed version.

(Inline Class) – Merges a class into another when it is too small or redundant.

(Extract Interface) – Creates a new interface from existing method(s) in a class.

(Move Method) – Relocates a method to a more appropriate class.

(Move Field) – Relocates a field to a more appropriate class.

(Replace Magic Number with Constant) – Replaces a literal number with a named constant.

(Encapsulate Field) – Change a field access from not private to private and create get and set methods.

**\*\*Instructions:\*\***

1. Begin your response with a bullet-point list of the refactoring type(s) applied.
2. Then, briefly justify each identified refactoring with reference to the specific code changes.
3. Only include refactorings from the list above.
4. Be concise but precise in your explanations.

Do not generate explanations unrelated to the given transformation.

⋮

You are an expert coding assistant specialized in software refactoring, with many years of experience analyzing code transformations.

You will be given the diffs of a commit:

- **\*\*Diffs:\*\***

diff --git

src/gr/uom/java/xmi/decomposition/VariableDeclaration.java

src/gr/uom/java/xmi/decomposition/VariableDeclaration.java

index 0029efa5c..f7be365c6 100644

```
---  
src/gr/uom/java/xmi/decomposition/VariableDeclaration.java  
+++  
src/gr/uom/java/xmi/decomposition/VariableDeclaration.java  
@@ -2,6 +2,7 @@ package gr.uom.java.xmi.decomposition;  
  
import java.util.ArrayList;  
import java.util.List;  
+import java.util.Set;  
  
import org.eclipse.jdt.core.dom.ASTNode;  
import org.eclipse.jdt.core.dom.Annotation;  
@@ -256,4 +257,10 @@ public class VariableDeclaration  
implements LocationInfoProvider, VariableDeclar  
    public VariableDeclaration getVariableDeclaration() {  
        return this;  
    }  
+  
+    +    boolean consistencyCheck(VariableReplacementAnalysis  
variableReplacementAnalysis, VariableDeclaration v2,  
Set<AbstractCodeMapping> set) {  
+        return  
!variableReplacementAnalysis.variableAppearsInExtractedMet  
hod(this, v2) &&  
+  
!variableReplacementAnalysis.variableAppearsInTheInitializer  
OfTheOtherVariable(this, v2) &&  
+  
!variableReplacementAnalysis.inconsistentVariableMapping(t  
his, v2, set);  
+    }  
}  
diff --git  
src/gr/uom/java/xmi/decomposition/VariableReplacementAnaly  
sis.java
```

```
src/gr/uom/java/xmi/decomposition/VariableReplacementAnaly  
sis.java  
index 31c83ffc1..9166a6e1f 100644  
---  
src/gr/uom/java/xmi/decomposition/VariableReplacementAnaly  
sis.java  
+++  
src/gr/uom/java/xmi/decomposition/VariableReplacementAnaly  
sis.java  
@@ -426,7 +426,7 @@ public class  
VariableReplacementAnalysis {  
    for(Replacement replacement :  
allConsistentVariableDeclarationRenames) {  
        VariableDeclarationReplacement vdReplacement  
= (VariableDeclarationReplacement)replacement;  
        Set<AbstractCodeMapping> set =  
variableDeclarationReplacementOccurrenceMap.get(vdRepla  
cement);  
        -            if((set.size() > 1 &&  
consistencyCheck(vdReplacement.getVariableDeclaration1(),  
vdReplacement.getVariableDeclaration2(), set)) ||  
        +            if((set.size() > 1 &&  
vdReplacement.getVariableDeclaration1().consistencyCheck(t  
his, vdReplacement.getVariableDeclaration2(), set)) ||  
                (set.size() == 1 &&  
replacementInLocalVariableDeclaration(vdReplacement.getVa  
riableNameReplacement(), set)) {  
            RenameVariableRefactoring ref = new  
RenameVariableRefactoring(vdReplacement.getVariableDecla  
ration1(), vdReplacement.getVariableDeclaration2(),  
vdReplacement.getOperation1(),  
vdReplacement.getOperation2(), set);  
  
if(!existsConflictingExtractVariableRefactoring(ref) &&  
!existsConflictingMergeVariableRefactoring(ref) &&  
!existsConflictingSplitVariableRefactoring(ref)) {
```

```

@@ -452,13 +452,13 @@ public class
VariableReplacementAnalysis {
    SimpleEntry<VariableDeclaration,
UMLOperation> v1 = getVariableDeclaration1(replacement);
    SimpleEntry<VariableDeclaration,
UMLOperation> v2 = getVariableDeclaration2(replacement);
    Set<AbstractCodeMapping> set =
replacementOccurrenceMap.get(replacement);
-           if((set.size() > 1 && v1 != null && v2 != null &&
consistencyCheck(v1.getKey(), v2.getKey(), set)) ||
+           if((set.size() > 1 && v1 != null && v2 != null &&
v1.getKey().consistencyCheck(this, v2.getKey(), set)) ||
potentialParameterRename(replacement, set) ||
v1 == null || v2 == null ||
(set.size() == 1 &&
replacementInLocalVariableDeclaration(replacement, set))) {
finalConsistentRenames.put(replacement,
set);
}
-           if(v1 != null && !v1.getKey().isParameter() && v2
!= null && v2.getKey().isParameter() &&
consistencyCheck(v1.getKey(), v2.getKey(), set) &&
+           if(v1 != null && !v1.getKey().isParameter() && v2
!= null && v2.getKey().isParameter() &&
v1.getKey().consistencyCheck(this, v2.getKey(), set) &&
!operation1.getParameterNameList().contains(v2.getKey().get
VariableName())) {
finalConsistentRenames.put(replacement,
set);
}
@@ -784,16 +784,10 @@ public class
VariableReplacementAnalysis {
    v1.equalVariableDeclarationType(v2) &&

```

```

!containsVariableDeclarationWithName(allVariableDeclaration
s1, v2.getVariableName()) &&

(!containsVariableDeclarationWithName(allVariableDeclaration
s2,
v1.getVariableName()) ||
operation2.loopWithVariables(v1.getVariableName(),
v2.getVariableName() != null) &&
-
            consistencyCheck(v1, v2, set);
+
            v1.consistencyCheck(this, v2, set);
}

-
    private boolean consistencyCheck(VariableDeclaration
v1, VariableDeclaration v2, Set<AbstractCodeMapping> set) {
-
        return !variableAppearsInExtractedMethod(v1, v2)
&&
-
        !variableAppearsInTheInitializerOfTheOtherVariable(v1, v2)
&&
-
        !inconsistentVariableMapping(v1, v2, set);
-
    }

-
    private boolean
variableAppearsInTheInitializerOfTheOtherVariable(VariableD
eclaration v1, VariableDeclaration v2) {
+
    boolean
variableAppearsInTheInitializerOfTheOtherVariable(VariableD
eclaration v1, VariableDeclaration v2) {
        if(v1.getInitializer() != null) {

if(v1.getInitializer().getString().equals(v2.getVariableName()))
{
    return true;
@@ -831,7 +825,7 @@ public class
VariableReplacementAnalysis {

```

```

        return false;
    }

    -  private boolean
inconsistentVariableMapping(VariableDeclaration v1,
VariableDeclaration
v2, Set<AbstractCodeMapping> set) {
    +  boolean
inconsistentVariableMapping(VariableDeclaration v1,
VariableDeclaration v2, Set<AbstractCodeMapping> set) {
        if(v1 != null && v2 != null) {
            for(AbstractCodeMapping mapping : mappings)
{
                List<VariableDeclaration>
variableDeclarations1 = mapping.getFra
gment1().getVariableDeclarations();
@@ -1040,7 +1034,7 @@ public class
VariableReplacementAnalysis {
        return null;
    }

    -  private boolean
variableAppearsInExtractedMethod(VariableDeclaration v1,
VariableDeclaration v2) {
    +  boolean
variableAppearsInExtractedMethod(VariableDeclaration v1,
VariableDeclaration v2) {
        if(v1 != null) {
            for(UMLOperationBodyMapper mapper :
childMappers) {
                for(AbstractCodeMapping mapping :
mapper.getMappings()) {

```

Your task is to identify which refactoring type(s) have been applied in transforming the original program into the new version. Use only the following list of predefined refactorings:

- (Add Method Parameter) – Introduces a new parameter to an existing method.
- (Remove Method Parameter) – Eliminates an existing parameter from a method signature.
- (Rename Method) – Changes the name of a method while preserving its behavior.
- (Rename Class) – Changes the name of a class without altering its structure.
- (Rename Package) – Changes the name of a package declaration.
- (Rename Field) – Changes the name of a class or instance variable.
- (Extract Class) – Moves a group of related fields and methods from an existing class into a newly created class.
- (Extract Superclass) – Creates a new superclass to encapsulate shared attributes and behavior from two or more existing classes.
- (Inline Method) – Replaces a method call with the method's body.
- (Pull Up Method) – Pull a method from a child class (subclass) in the original code to its parent class (superclass) in the transformed version.
- (Push Down Method) – Push a method from a parent class (superclass) in the original code to one or more of its child classes (subclasses) in the transformed version.
- (Pull Up Field) – Pull a field from a child class (subclass) in the original code to its parent class (superclass) in the transformed version.
- (Push Down Field) – Push a field from a parent class (superclass) in the original code to one or more of its child classes (subclasses) in the transformed version.
- (Inline Class) – Merges a class into another when it is too small or redundant.
- (Extract Interface) – Creates a new interface from existing method(s) in a class.

(Move Method) – Relocates a method to a more appropriate class.

(Move Field) – Relocates a field to a more appropriate class.

(Replace Magic Number with Constant) – Replaces a literal number with a named constant.

(Encapsulate Field) – Change a field access from not private to private and create get and set methods.

**\*\*Instructions:\*\***

1. Begin your response with a bullet-point list of the refactoring type(s) applied.

2. Then, briefly justify each identified refactoring with reference to the specific code changes.

3. Only include refactorings from the list above.

4. Be concise but precise in your explanations.

Do not generate explanations unrelated to the given transformation.

⋮

You are an expert coding assistant specialized in software refactoring, with many years of experience analyzing code transformations.

You will be given the diffs of a commit:

- **\*\*Diffs:\*\***

```
diff --git src/main/java/com/google/maps/PendingResult.java
src/main/java/com/google/maps/PendingResult.java
index 1c9acc28..7df5525e 100644
--- src/main/java/com/google/maps/PendingResult.java
+++ src/main/java/com/google/maps/PendingResult.java
@@ -16,6 +16,10 @@
```

```
package com.google.maps;

import com.google.maps.errors.ApiException;
+import com.google.maps.internal.OkHttpPendingResult;
+
+import okhttp3.Response;
+
import java.io.IOException;

/**
@@ -77,5 +81,11 @@ public interface PendingResult<T> {
     * @param e The exception describing the failure.
     */
    void onFailure(Throwable e);

+
+    public default boolean
shouldRetry(OkHttpPendingResult okHttpPendingResult,
Response response) {
+
        return
OkHttpPendingResult.RETRY_ERROR_CODES.contains(respon
se.code())
+
        && okHttpPendingResult.cumulativeSleepTime <
okHttpPendingResult.errorTimeOut
+
        && (okHttpPendingResult.maxRetries == null ||
okHttpPendingResult.retryCounter <
okHttpPendingResult.maxRetries);
+
    }
}
}

diff --git
src/main/java/com/google/maps/internal/OkHttpPendingResult
.java
src/main/java/com/google/maps/internal/OkHttpPendingResult
.java
index a250a959..6271656f 100644
```

```
---  
src/main/java/com/google/maps/internal/OkHttpPendingResult  
.java  
+++  
src/main/java/com/google/maps/internal/OkHttpPendingResult  
.java  
@@ -65,18 +65,18 @@ public class OkHttpPendingResult<T, R  
extends ApiResponse<T>>  
    private final OkHttpClient client;  
    private final Class<R> responseClass;  
    private final FieldNamingPolicy fieldNamingPolicy;  
-    private final Integer maxRetries;  
+    public final Integer maxRetries;  
    private final RequestMetrics metrics;  
  
    private Call call;  
    private Callback<T> callback;  
-    private long errorTimeOut;  
-    private int retryCounter = 0;  
-    private long cumulativeSleepTime = 0;  
+    public long errorTimeOut;  
+    public int retryCounter = 0;  
+    public long cumulativeSleepTime = 0;  
    private ExceptionsAllowedToRetry  
exceptionsAllowedToRetry;  
  
    private static final Logger LOG =  
LoggerFactory.getLogger(OkHttpPendingResult.class.getName()  
);  
-    private static final List<Integer> RETRY_ERROR_CODES =  
Arrays.asList(500, 503, 504);  
+    public static final List<Integer> RETRY_ERROR_CODES =  
Arrays.asList(500, 503, 504);  
  
/**
```

```

 * @param request HTTP request to execute.
@@ -238,7 +238,7 @@ public class OkHttpPendingResult<T, R
extends ApiResponse<T>>
    @SuppressWarnings("unchecked")
    private T parseResponseInternal(OkHttpPendingResult<T, R>
request, Response response)
        throws ApiException, InterruptedException, IOException {
-    if (shouldRetry(response)) {
+    if (callback.shouldRetry(this, response)) {
        // since we are retrying the request we must close the
        response.close();
    }
@@ -327,12 +327,6 @@ public class OkHttpPendingResult<T, R
extends ApiResponse<T>>
    return this.await();
}

-    private boolean shouldRetry(Response response) {
-        return RETRY_ERROR_CODES.contains(response.code())
-            && cumulativeSleepTime < errorTimeOut
-            && (maxRetries == null || retryCounter < maxRetries);
-    }
-
-    private boolean shouldRetry(ApiException exception) {
-        return
exceptionsAllowedToRetry.contains(exception.getClass())
            && cumulativeSleepTime < errorTimeOut

```

Your task is to identify which refactoring type(s) have been applied in transforming the original program into the new version. Use only the following list of predefined refactorings:

(Add Method Parameter) – Introduces a new parameter to an existing method.

- (Remove Method Parameter) – Eliminates an existing parameter from a method signature.
- (Rename Method) – Changes the name of a method while preserving its behavior.
- (Rename Class) – Changes the name of a class without altering its structure.
- (Rename Package) – Changes the name of a package declaration.
- (Rename Field) – Changes the name of a class or instance variable.
- (Extract Class) – Moves a group of related fields and methods from an existing class into a newly created class.
- (Extract Superclass) – Creates a new superclass to encapsulate shared attributes and behavior from two or more existing classes.
- (Inline Method) – Replaces a method call with the method's body.
- (Pull Up Method) – Pull a method from a child class (subclass) in the original code to its parent class (superclass) in the transformed version.
- (Push Down Method) – Push a method from a parent class (superclass) in the original code to one or more of its child classes (subclasses) in the transformed version.
- (Pull Up Field) – Pull a field from a child class (subclass) in the original code to its parent class (superclass) in the transformed version.
- (Push Down Field) – Push a field from a parent class (superclass) in the original code to one or more of its child classes (subclasses) in the transformed version.
- (Inline Class) – Merges a class into another when it is too small or redundant.
- (Extract Interface) – Creates a new interface from existing method(s) in a class.
- (Move Method) – Relocates a method to a more appropriate class.
- (Move Field) – Relocates a field to a more appropriate class.

(Replace Magic Number with Constant) – Replaces a literal number with a named constant.

(Encapsulate Field) – Change a field access from not private to private and create get and set methods.

**\*\*Instructions:\*\***

1. Begin your response with a bullet-point list of the refactoring type(s) applied.
2. Then, briefly justify each identified refactorng with reference to the specific code changes.
3. Only include refactorings from the list above.
4. Be concise but precise in your explanations.

Do not generate explanations unrelated to the given transformation.

⋮

You are an expert coding assistant specialized in software refactoring, with many years of experience analyzing code transformations.

You will be given the diffs of a commit:

- **\*\*Diffs:\*\***

```
diff --git spring-boot-project/spring-boot-
actuator/src/main/java/org/springframework/boot/actuate/end
point/jmx/JmxOperationResponseMapper.java spring-boot-
project/spring-boot-
actuator/src/main/java/org/springframework/boot/actuate/end
point/jmx/JmxOperationResponseMapper.java
index 1e291e1a50..a45ebbd356 100644
--- spring-boot-project/spring-boot-
actuator/src/main/java/org/springframework/boot/actuate/end
```

```
point/jmx/JmxOperationResponseMapper.java
+++ spring-boot-project/spring-boot-
actuator/src/main/java/org/springframework/boot/actuate/end
point/jmx/JmxOperationResponseMapper.java
@@ -38,4 +38,8 @@ public interface
JmxOperationResponseMapper {
*/
Object mapResponse(Object response);

+ default String getType(MBeanInfoFactory
mBeanInfoFactory, Class<?> outputType) {
+ return mapResponseType(outputType).getName();
+
+
}
diff --git spring-boot-project/spring-boot-
actuator/src/main/java/org/springframework/boot/actuate/end
point/jmx/MBeanInfoFactory.java spring-boot-project/spring-
boot-
actuator/src/main/java/org/springframework/boot/actuate/end
poi
nt/jmx/MBeanInfoFactory.java
index 097fe109b5..e53345abc9 100644
--- spring-boot-project/spring-boot-
actuator/src/main/java/org/springframework/boot/actuate/end
point/jmx/MBeanInfo
Factory.java
+++ spring-boot-project/spring-boot-
actuator/src/main/java/org/springframework/boot/actuate/end
point/jmx/MBeanInfoFactory.java
@@ -69,7 +69,7 @@ class MBeanInfoFactory {
String name = operation.getName();
String description = operation.getDescription();
MBeanParameterInfo[] signature =
getSignature(operation.getParameters());
- String type = getType(operation.getOutputType());
```

```

+ String type = responseMapper.getType(this,
operation.getOutputType());
int impact = getImpact(operation.getType());
return new ModelMBeanOperationInfo(name, description,
signature, type, impact);
}

@@ -92,8 +92,4 @@ class MBeanInfoFactory {
return MBeanOperationInfo.UNKNOWN;
}

-
```

- private String getType(Class<?> outputType) {  
- return  
this.responseMapper.mapResponseType(outputType).getName();  
- }  
-  
}

Your task is to identify which refactoring type(s) have been applied in transforming the original program into the new version. Use only the following list of predefined refactorings:

(Add Method Parameter) – Introduces a new parameter to an existing method.

(Remove Method Parameter) – Eliminates an existing parameter from a method signature.

(Rename Method) – Changes the name of a method while preserving its behavior.

(Rename Class) – Changes the name of a class without altering its structure.

(Rename Package) – Changes the name of a package declaration.

(Rename Field) – Changes the name of a class or instance variable.

(Extract Class) – Moves a group of related fields and methods from an existing class into a newly created class.

(Extract Superclass) – Creates a new superclass to encapsulate shared attributes and behavior from two or more existing classes.

(Inline Method) – Replaces a method call with the method's body.

(Pull Up Method) – Pull a method from a child class (subclass) in the original code to its parent class (superclass) in the transformed version.

(Push Down Method) – Push a method from a parent class (superclass) in the original code to one or more of its child classes (subclasses) in the transformed version.

(Pull Up Field) – Pull a field from a child class (subclass) in the original code to its parent class (superclass) in the transformed version.

(Push Down Field) – Push a field from a parent class (superclass) in the original code to one or more of its child classes (subclasses) in the transformed version.

(Inline Class) – Merges a class into another when it is too small or redundant.

(Extract Interface) – Creates a new interface from existing method(s) in a class.

(Move Method) – Relocates a method to a more appropriate class.

(Move Field) – Relocates a field to a more appropriate class.

(Replace Magic Number with Constant) – Replaces a literal number with a named constant.

(Encapsulate Field) – Change a field access from not private to private and create get and set methods.

**\*\*Instructions:\*\***

1. Begin your response with a bullet-point list of the refactoring type(s) applied.
2. Then, briefly justify each identified refactoring with reference to the specific code changes.
3. Only include refactorings from the list above.
4. Be concise but precise in your explanations.

Do not generate explanations unrelated to the given transformation.

⋮

You are an expert coding assistant specialized in software refactoring, with many years of experience analyzing code transformations.

You will be given the diffs of a commit:

```
- **Diffs:**  
diff --git  
src/gr/uom/java/xmi/decomposition/CompositeStatementObje  
ct.java  
src/gr/uom/java/xmi/decomposition/CompositeStatementObje  
ct.java  
index f43a86952..c574f4e55 100644  
---  
src/gr/uom/java/xmi/decomposition/CompositeStatementObje  
ct.java  
+++  
src/gr/uom/java/xmi/decomposition/CompositeStatementObje  
ct.java  
@@ -531,4 +531,8 @@ public class  
CompositeStatementObject extends AbstractStatement {  
}  
return null;  
}  
+^M  
+    public void setFinallyClause(TryStatementObject  
tryStatementObject) {^M  
+        tryStatementObject.finallyClause = this;^M
```

```

+      }^M
}
diff --git
src/gr/uom/java/xmi/decomposition/OperationBody.java
src/gr/uom/java/xmi/decomposition/OperationBody.java
index e05112897..1fc97b031 100644
--- src/gr/uom/java/xmi/decomposition/OperationBody.java
+++ src/gr/uom/java/xmi/decomposition/OperationBody.java
@@ -250,7 +250,7 @@ public class OperationBody {
        Block finallyBlock = tryStatement.getFinally();
        if(finallyBlock != null) {
            CompositeStatementObject
finallyClauseStatementObject = new
CompositeStatementObject(cu, filePath, finallyBlock,
parent.getDepth()+1, CodeElementType.FINALLY_BLOCK);
-
        child.setFinallyClause(finallyClauseStatementObject);
+
        finallyClauseStatementObject.setFinallyClause(child);^M

parent.addStatement(finallyClauseStatementObject);
        List<Statement> blockStatements =
finallyBlock.statements();
        for(Statement blockStatement :
blockStatements) {
diff --git
src/gr/uom/java/xmi/decomposition/TryStatementObject.java
src/gr/uom/java/xmi/decomposition/TryStatementObject.java
index 346d12ef5..039724031 100644
---
src/gr/uom/java/xmi/decomposition/TryStatementObject.java
+++
src/gr/uom/java/xmi/decomposition/TryStatementObject.java
@@ -10,7 +10,7 @@ import
gr.uom.java.xmi.LocationInfo.CodeElementType;

```

```

public class TryStatementObject extends
CompositeStatementObject {
    private List<CompositeStatementObject> catchClauses;
-    private CompositeStatementObject finallyClause;
+    CompositeStatementObject finallyClause;

    public TryStatementObject(CompilationUnit cu, String
filePath, Statement statement, int
depth) {
        super(cu, filePath, statement, depth,
CodeElementType.TRY_STATEMENT);
@@ -25,10 +25,6 @@ public class TryStatementObject
extends CompositeStatementObject {
        return catchClauses;
    }

-    public void setFinallyClause(CompositeStatementObject
finallyClause) {
-        this.finallyClause = finallyClause;
-    }
-
-    public CompositeStatementObject getFinallyClause() {
-        return finallyClause;
-    }
}

```

Your task is to identify which refactoring type(s) have been applied in transforming the original program into the new version. Use only the following list of predefined refactorings:

(Add Method Parameter) – Introduces a new parameter to an existing method.

(Remove Method Parameter) – Eliminates an existing parameter from a method signature.

(Rename Method) – Changes the name of a method while preserving its behavior.

(Rename Class) – Changes the name of a class without altering its structure.

(Rename Package) – Changes the name of a package declaration.

(Rename Field) – Changes the name of a class or instance variable.

(Extract Class) – Moves a group of related fields and methods from an existing class into a newly created class.

(Extract Superclass) – Creates a new superclass to encapsulate shared attributes and behavior from two or more existing classes.

(Inline Method) – Replaces a method call with the method's body.

(Pull Up Method) – Pull a method from a child class (subclass) in the original code to its parent class (superclass) in the transformed version.

(Push Down Method) – Push a method from a parent class (superclass) in the original code to one or more of its child classes (subclasses) in the transformed version.

(Pull Up Field) – Pull a field from a child class (subclass) in the original code to its parent class (superclass) in the transformed version.

(Push Down Field) – Push a field from a parent class (superclass) in the original code to one or more of its child classes (subclasses) in the transformed version.

(Inline Class) – Merges a class into another when it is too small or redundant.

(Extract Interface) – Creates a new interface from existing method(s) in a class.

(Move Method) – Relocates a method to a more appropriate class.

(Move Field) – Relocates a field to a more appropriate class.

(Replace Magic Number with Constant) – Replaces a literal number with a named constant.

(Encapsulate Field) – Change a field access from not private to private and create get and set methods.

**\*\*Instructions:\*\***

1. Begin your response with a bullet-point list of the refactoring type(s) applied.
2. Then, briefly justify each identified refactoring with reference to the specific code changes.
3. Only include refactorings from the list above.
4. Be concise but precise in your explanations.

Do not generate explanations unrelated to the given transformation.

⋮

You are an expert coding assistant specialized in software refactoring, with many years of experience analyzing code transformations.

You will be given the diffs of a commit:

- **\*\*Diffs:\*\***

```
diff --git spring-boot-project/spring-boot-actuator-
autoconfigure/src/main/java/org/springframework/boot/actuat-
e/autoconfigure/web/reactive/ReactiveManagementChildCont-
extConfiguration.java spring-boot-project/spring-boot-
actuator-
autoconfigure/src/main/java/org/springframework/boot/actuat-
e/autoconfigure/web/reactive/ReactiveManagementChildCont-
extConfiguration.java
index d933d8f4ef..ca2bbb7316 100644
--- spring-boot-project/spring-boot-actuator-
autoconfigure/src/main/java/org/springframework/boot/actuat-
e/autoconfigure/web/reactive/ReactiveManagementChildCont-
extConfiguration.java
```

```
+++ spring-boot-project/spring-boot-actuator-
autoconfigure/src/main/java/org/springframework/boot/actuat-
e/autoconfigure/web/reactive/ReactiveManagementChildCont-
extConfiguration.java
@@ -16,12 +16,17 @@
package
org.springframework.boot.actuate.autoconfigure.web.reactive
;

+import java.util.Collections;
+
+import org.springframework.beans.factory.BeanFactoryUtils;
import
org.springframework.beans.factory.ListableBeanFactory;
import
org.springframework.boot.actuate.autoconfigure.web.ManagementContextConfiguration;
import
org.springframework.boot.actuate.autoconfigure.web.ManagementContextType;
+import
org.springframework.boot.actuate.autoconfigure.web.server.ManagementServerProperties;
import
org.springframework.boot.actuate.autoconfigure.web.server.ManagementWebServerFactoryCustomizer;
import
org.springframework.boot.autoconfigure.condition.Conditiona
lOnWebApplication;
import
org.springframework.boot.autoconfigure.condition.Conditiona
lOnWebApplication.Type;
+import
org.springframework.boot.autoconfigure.web.ServerPropertie
s;
```

```
import
org.springframework.boot.autoconfigure.web.embedded.Jett
yWebServerFactoryCustomizer;
import
org.springframework.boot.autoconfigure.web.embedded.Nett
yWebServerFactoryCustomizer;
import
org.springframework.boot.autoconfigure.web.embedded.Tom
catWebServerFactoryCustomizer;
@@ -69,6 +74,21 @@ public class
ReactiveManagementChildContextConfiguration {
UndertowWebServerFactoryCustomizer.class,
NettyWebServerFactoryCustomizer.class);
}

+ @Override
+ public final void
customize(ConfigurableReactiveWebServerFactory factory) {
+ ManagementServerProperties
managementServerProperties = BeanFactoryUtils
+ .beanOfTypeIncludingAncestors(this.beanFactory,
ManagementServerProperties.class);
+ // Customize as per the parent context first (so e.g. the
access logs go to
+ // the same place)
+ customizeSameAsParentContext(factory);
+ // Then reset the error pages
+ factory.setErrorPages(Collections.emptySet());
+ // and add the management-specific bits
+ ServerProperties serverProperties =
BeanFactoryUtils.beanOfTypeIncludingAncestors(this.beanFa
ctory,
+ ServerProperties.class);
+ customize(factory, managementServerProperties,
serverProperties);
+ }
```

```
+  
}  
}  
  
diff --git spring-boot-project/spring-boot-actuator-  
autoconfigure/src/main/java/org/springframework/boot/actuat  
e/autoconfigure/web/server/ManagementWebServerFactoryC  
ustomizer.java spring-boot-project/spring-boot-actuator-  
autoconfigure/src/main/java/org/springframework/boot/actuat  
e/autoconfigure/web/server/ManagementWebServerFactoryC  
ustomizer.java  
index 4b0e3be415..71fd8486f3 100644  
--- spring-boot-project/spring-boot-actuator-  
autoconfigure/src/main/java/org/springframework/boot/actuat  
e/autoconfigure/web/server/ManagementWebServerFactoryC  
ustomizer.java  
+++ spring-boot-project/spring-boot-actuator-  
autoconfigure/src/main/java/org/springframework/boot/actuat  
e/autoconfigure/web/server/ManagementWebServerFactoryC  
ustomizer.java  
@@ -17,7 +17,6 @@  
package  
org.springframework.boot.actuate.autoconfigure.web.server;  
  
import java.util.Arrays;  
-import java.util.Collections;  
import java.util.List;  
import java.util.Objects;  
import java.util.stream.Collectors;  
@@ -44,7 +43,7 @@ import  
org.springframework.core.Ordered;  
public abstract class  
ManagementWebServerFactoryCustomizer<T extends  
ConfigurableWebServerFactory>  
implements WebServerFactoryCustomizer<T>, Ordered {  
  
- private final ListableBeanFactory beanFactory;
```

```
+ protected final ListableBeanFactory beanFactory;

private final Class<? extends WebServerFactoryCustomizer<?>>[] customizerClasses;

@@ -61,22 +60,7 @@ public abstract class
ManagementWebServerFactoryCustomizer<T extends
Configurable
return 0;
}

- @Override
- public final void customize(T factory) {
- ManagementServerProperties
managementServerProperties = BeanFactoryUtils
- .beanOfTypeIncludingAncestors(this.beanFactory,
ManagementServerProperties.class);- // Customize as per the
parent context first (so e.g. the access logs go to
- // the same place)
- customizeSameAsParentContext(factory);
- // Then reset the error pages
- factory.setErrorPages(Collections.emptySet());
- // and add the management-specific bits
- ServerProperties serverProperties =
BeanFactoryUtils.beanOfTypeIncludingAncestors(this.beanFa
ctory,
- ServerProperties.class);
- customize(factory, managementServerProperties,
serverProperties);
- }
-
- private void customizeSameAsParentContext(T factory) {
+ protected void customizeSameAsParentContext(T factory) {
List<WebServerFactoryCustomizer<?>> customizers =
Arrays.stream(this.customizerClasses).map(this::getCustomiz
er)
```

```
.filter(Objects::nonNull).collect(Collectors.toList()));

invokeCustomizers(factory, customizers);

diff --git spring-boot-project/spring-boot-actuator-
autoconfigure/src/main/java/org/springframework/boot/actuat
e/autoconfigure/web/servlet/ServletManagementChildContext
Configuration.java spring-boot-project/spring-boot-actuator-
autoconfigure/src/main/java/org/springframework/boot/actuat
e/autoconfigure/web/servlet/ServletManagementChildContext
Configuration.java

index 15a47bf321..f1c0cebb94 100644
--- spring-boot-project/spring-boot-actuator-
autoconfigure/src/main/java/org/springframework/boot/actuat
e/autoconf
igure/web/servlet/ServletManagementChildContextConfigurati
on.java
+++ spring-boot-project/spring-boot-actuator-
autoconfigure/src/main/java/org/springframework/boot/actuat
e/autoconfigure/web/servlet/ServletManagementChildContext
Configuration.java
@@ -17,6 +17,7 @@
package

org.springframework.boot.actuate.autoconfigure.web.servlet;

import java.io.File;
+import java.util.Collections;

import javax.servlet.Filter;

@@ -28,6 +29,7 @@ import
org.eclipse.jetty.server.RequestLogWriter;
import org.eclipse.jetty.server.Server;

import org.springframework.beans.factory.BeanFactory;
+import org.springframework.beans.factory.BeanFactoryUtils;
import
org.springframework.beans.factory.HierarchicalBeanFactory;
```

```
import
org.springframework.beans.factory.ListableBeanFactory;
import
org.springframework.boot.actuate.autoconfigure.web.ManagementContextConfiguration;
@@ -126,6 +128,21 @@ class
ServletManagementChildContextConfiguration {
    webServerFactory.setContextPath(managementServerProperties.getServlet().getContextPath()); }

+ @Override
+ public final void
customize(ConfigurableServletWebServerFactory factory) {
+ ManagementServerProperties
managementServerProperties = BeanFactoryUtils
+ .beanOfTypeIncludingAncestors(this.beanFactory,
ManagementServerProperties
.class);
+ // Customize as per the parent context first (so e.g. the
access logs go to
+ // the same place)
+ customizeSameAsParentContext(factory);
+ // Then reset the error pages
+ factory.setErrorPages(Collections.emptySet());
+ // and add the management-specific bits
+ ServerProperties serverProperties =
BeanFactoryUtils.beanOfTypeIncludingAncestors(this.beanFactory,
+ ServerProperties.class);
+ customize(factory, managementServerProperties,
serverProperties);
+ }
+
}
```

```
abstract static class AccessLogCustomizer implements  
Ordered {
```

Your task is to identify which refactoring type(s) have been applied in transforming the original program into the new version. Use only the following list of predefined refactorings:

(Add Method Parameter) – Introduces a new parameter to an existing method.

(Remove Method Parameter) – Eliminates an existing parameter from a method signature.

(Rename Method) – Changes the name of a method while preserving its behavior.

(Rename Class) – Changes the name of a class without altering its structure.

(Rename Package) – Changes the name of a package declaration.

(Rename Field) – Changes the name of a class or instance variable.

(Extract Class) – Moves a group of related fields and methods from an existing class into a newly created class.

(Extract Superclass) – Creates a new superclass to encapsulate shared attributes and behavior from two or more existing classes.

(Inline Method) – Replaces a method call with the method's body.

(Pull Up Method) – Pull a method from a child class (subclass) in the original code to its parent class (superclass) in the transformed version.

(Push Down Method) – Push a method from a parent class (superclass) in the original code to one or more of its child classes (subclasses) in the transformed version.

(Pull Up Field) – Pull a field from a child class (subclass) in the original code to its parent class (superclass) in the transformed version.

(Push Down Field) – Push a field from a parent class (superclass) in the original code to one or more of its child classes (subclasses) in the transformed version.

(Inline Class) – Merges a class into another when it is too small or redundant.

(Extract Interface) – Creates a new interface from existing method(s) in a class.

(Move Method) – Relocates a method to a more appropriate class.

(Move Field) – Relocates a field to a more appropriate class.

(Replace Magic Number with Constant) – Replaces a literal number with a named constant.

(Encapsulate Field) – Change a field access from not private to private and create get and set methods.

**\*\*Instructions:\*\***

1. Begin your response with a bullet-point list of the refactoring type(s) applied.
2. Then, briefly justify each identified refactoring with reference to the specific code changes.
3. Only include refactorings from the list above.
4. Be concise but precise in your explanations.

Do not generate explanations unrelated to the given transformation.

⋮

You are an expert coding assistant specialized in software refactoring, with many years of experience analyzing code transformations.

You will be given the diffs of a commit:

```
- ***Diffs:**  
diff --git src-  
test/org/refactoringminer/utils/RefactoringCollector.java src-  
test/org/refactoringminer/utils/RefactoringCollector.java  
index 54f06053f..2b2ce9ba9 100644  
--- src-  
test/org/refactoringminer/utils/RefactoringCollector.java  
+++ src-  
test/org/refactoringminer/utils/RefactoringCollector.java  
@@ -8,7 +8,7 @@ import  
org.refactoringminer.api.RefactoringHandler;  
import gr.uom.java.xmi.diff.ExtractOperationRefactoring;  
import gr.uom.java.xmi.diff.ExtractSuperclassRefactoring;  
import gr.uom.java.xmi.diff.InlineOperationRefactoring;  
-import gr.uom.java.xmi.diff.MoveAttributeRefactoring;  
+import  
gr.uom.java.xmi.diff.MoveAttributeRefactoring_RENAMED;^M  
import gr.uom.java.xmi.diff.MoveClassRefactoring;  
import gr.uom.java.xmi.diff.MoveOperationRefactoring;  
import gr.uom.java.xmi.diff.RenameClassRefactoring;  
@@ -46,8 +46,8 @@ public class RefactoringCollector  
extends RefactoringHandler {  
} else if (r instanceof InlineOperationRefactoring) {  
    InlineOperationRefactoring ref =  
(InlineOperationRefactoring) r;  
    rs.add(new  
RefactoringRelationship(r.getRefactoringType(),  
ref.getInlinedOperation().getKey(),  
ref.getTargetOperationAfterInline().getKey()));  
-    } else if (r instanceof MoveAttributeRefactoring) {  
-        MoveAttributeRefactoring ref =  
(MoveAttributeRefactoring) r;  
+    } else if (r instanceof  
MoveAttributeRefactoring_RENAMED) {^M  
+        MoveAttributeRefactoring_RENAMED ref =  
(MoveAttributeRefactoring_RENAMED) r;^M
```

```
String attrName = ref.getMovedAttribute().getName();
rs.add(new
RefactoringRelationship(r.getRefactoringType(),
ref.getSourceClassName() + "#" + attrName,
ref.getTargetClassName() + "#" + attrName));
} else {
diff --git
src/gr/uom/java/xmi/diff/MoveAndRenameAttributeRefactoring.
java
src/gr/uom/java/xmi/diff/MoveAndRenameAttributeRefactoring.
java
index Oaa2fc932..fb86f6584 100644
---
src/gr/uom/java/xmi/diff/MoveAndRenameAttributeRefactoring.
java
+++  

src/gr/uom/java/xmi/diff/MoveAndRenameAttributeRefactoring.
java
@@ -8,7 +8,7 @@ import
org.refactoringminer.api.RefactoringType;

import gr.uom.java.xmi.UMLAttribute;

-public class MoveAndRenameAttributeRefactoring extends
MoveAttributeRefactoring {
+public class MoveAndRenameAttributeRefactoring extends
MoveAttributeRefactoring_RENAMED {
    private Set<CandidateAttributeRefactoring>
attributeRenames;

    public
MoveAndRenameAttributeRefactoring(UMLAttribute
originalAttribute, UMLAttribute movedAttribute,
diff --git
src/gr/uom/java/xmi/diff/MoveAttributeRefactoring.java
```

```
src/gr/uom/java/xmi/diff/MoveAttributeRefactoring_RENAMED.java
similarity index 89%
rename from
src/gr/uom/java/xmi/diff/MoveAttributeRefactoring.java
rename to
src/gr/uom/java/xmi/diff/MoveAttributeRefactoring_RENAMED.java
index cecbf314e..872718462 100644
--- src/gr/uom/java/xmi/diff/MoveAttributeRefactoring.java
+++ src/gr/uom/java/xmi/diff/MoveAttributeRefactoring_RENAMED.java
@@ -11,12 +11,12 @@ import
org.refactoringminer.api.RefactoringType;

import gr.uom.java.xmi.UMLAttribute;

-public class MoveAttributeRefactoring implements
Refactoring {
+public class MoveAttributeRefactoring_RENAMED
implements Refactoring {^M
    protected UMLAttribute originalAttribute;
    protected UMLAttribute movedAttribute;
    private volatile int hashCode = 0;

-    public MoveAttributeRefactoring(UMLAttribute
originalAttribute, UMLAttribute movedAttribute) {
+    public
MoveAttributeRefactoring_RENAMED(UMLAttribute
originalAttribute, UMLAttribute movedAttribute) {^M
        this.originalAttribute = originalAttribute;
        this.movedAttribute = movedAttribute;
    }
@@ -77,8 +77,8 @@ public class MoveAttributeRefactoring
implements Refactoring {
```

```
        return true;
    }

-        if(o instanceof MoveAttributeRefactoring) {
-            MoveAttributeRefactoring other =
(MoveAttributeRefactoring)o;
+        if(o instanceof
MoveAttributeRefactoring_RENAMED) {^M
+            MoveAttributeRefactoring_RENAMED other =
(MoveAttributeRefactoring_RENAMED)o;^M
            return
this.originalAttribute.equals(other.originalAttribute) &&

this.movedAttribute.equals(other.movedAttribute) &&

this.getSourceClassName().equals(other.getSourceClassNam
e()) &&
diff --git
src/gr/uom/java/xmi/diff/MoveOperationRefactoring.java
src/gr/uom/java/xmi/diff/MoveOperationRefactoring.java
index 16d265e9c..e7c0eff0d 100644
--- src/gr/uom/java/xmi/diff/MoveOperationRefactoring.java
+++ src/gr/uom/java/xmi/diff/MoveOperationRefactoring.java
@@ -86,7 +86,7 @@ public class MoveOperationRefactoring
implements Refactoring {
        return movedOperation.codeRange();
    }

-    public boolean
compatibleWith(MoveAttributeRefactoring ref) {
+    public boolean
compatibleWith(MoveAttributeRefactoring_RENAMED ref) {^M

if(ref.getMovedAttribute().getClassName().equals(this.moved
Operation.getClassName())) &&
```

```
ref.getOriginalAttribute().getClassName().equals(this.original
Operation.getClassName())) {

    List<String> originalOperationVariables =
this.originalOperation.getAllVariables();
diff --git
src/gr/uom/java/xmi/diff/PullUpAttributeRefactoring.java
src/gr/uom/java/xmi/diff/PullUpAttributeRefactoring.java
index Od5882707..82c38cb33 100644
--- src/gr/uom/java/xmi/diff/PullUpAttributeRefactoring.java
+++ src/gr/uom/java/xmi/diff/PullUpAttributeRefactoring.java
@@ -7,7 +7,7 @@ import
org.refactoringminer.api.RefactoringType;

import gr.uom.java.xmi.UMLAttribute;

-public class PullUpAttributeRefactoring extends
MoveAttributeRefactoring {
+public class PullUpAttributeRefactoring extends
MoveAttributeRefactoring_RENAMED {^M

    public PullUpAttributeRefactoring(UMLAttribute
originalAttribute, UMLAttribute movedAttribute) {
        super(originalAttribute, movedAttribute);
diff --git
src/gr/uom/java/xmi/diff/PushDownAttributeRefactoring.java
src/gr/uom/java/xmi/diff/PushDownAttributeRefactoring.java
index c24cf1549..5bb2ead15 100644
---

src/gr/uom/java/xmi/diff/PushDownAttributeRefactoring.java
+++
src/gr/uom/java/xmi/diff/PushDownAttributeRefactoring.java
@@ -7,7 +7,7 @@ import
org.refactoringminer.api.RefactoringType;

import gr.uom.java.xmi.UMLAttribute;
```

```
-public class PushDownAttributeRefactoring extends
MoveAttributeRefactoring {
+public class PushDownAttributeRefactoring extends
MoveAttributeRefactoring_RENAMED {^M

    public PushDownAttributeRefactoring(UMLAttribute
originalAttribute, UMLAttribute movedAttribute) {
        super(originalAttribute, movedAttribute);
diff --git
src/gr/uom/java/xmi/diff/ReplaceAttributeRefactoring.java
src/gr/uom/java/xmi/diff/ReplaceAttributeRefactoring.java
index 48b7497d0..58be97d5a 100644
--- src/gr/uom/java/xmi/diff/ReplaceAttributeRefactoring.java
+++ src/gr/uom/java/xmi/diff/ReplaceAttributeRefactoring.java
@@ -8,7 +8,7 @@ import
org.refactoringminer.api.RefactoringType;

import gr.uom.java.xmi.UMLAttribute;

-public class ReplaceAttributeRefactoring extends
MoveAttributeRefactoring {
+public class ReplaceAttributeRefactoring extends
MoveAttributeRefactoring_RENAMED {
    private Set<CandidateAttributeRefactoring>
attributeRenames;

    public ReplaceAttributeRefactoring(UMLAttribute
originalAttribute, UMLAttribute movedAttribute,
diff --git src/gr/uom/java/xmi/diff/UMLModelDiff.java
src/gr/uom/java/xmi/diff/UMLModelDiff.java
index
c975c153c..2c3326e23 100644
--- src/gr/uom/java/xmi/diff/UMLModelDiff.java
+++ src/gr/uom/java/xmi/diff/UMLModelDiff.java
@@ -544,7 +544,7 @@ public class UMLModelDiff {
    return addedRealizations;
```

```
}

- private List<MoveAttributeRefactoring>
checkForAttributeMovesIncludingRemovedClasses() {
+ private List<MoveAttributeRefactoring_RENAMED>
checkForAttributeMovesIncludingRemovedClasses() {^M
    List<UMLAttribute> addedAttributes =
getAddedAttributesInCommonClasses();
/*for(UMLClass addedClass : addedClasses) {
    addedAttributes.addAll(addedClass.getAttributes());
@@ -556,7 +556,7 @@ public class UMLModelDiff {
    return checkForAttributeMoves(addedAttributes,
removedAttributes);
}
}

- private List<MoveAttributeRefactoring>
checkForAttributeMovesIncludingAddedClasses() {
+ private List<MoveAttributeRefactoring_RENAMED>
checkForAttributeMovesIncludingAddedClasses() {^M
    List<UMLAttribute> addedAttributes =
getAddedAttributesInCommonClasses();
for(UMLClass addedClass : addedClasses) {
    addedAttributes.addAll(addedClass.getAttributes());
@@ -568,13 +568,13 @@ public class UMLModelDiff {
    return checkForAttributeMoves(addedAttributes,
removedAttributes);
}
}

- private List<MoveAttributeRefactoring>
checkForAttributeMovesBetweenCommonClasses() {
+ private List<MoveAttributeRefactoring_RENAMED>
checkForAttributeMovesBetweenCommonClasses()
{^M
    List<UMLAttribute> addedAttributes =
getAddedAttributesInCommonClasses();
```

```

        List<UMLAttribute> removedAttributes =
getRemovedAttributesInCommonClasses();
        return checkForAttributeMoves(addedAttributes,
removedAttributes);
    }

- private List<MoveAttributeRefactoring>
checkForAttributeMovesBetweenRemovedAndAddedClasses(
)
{
+ private List<MoveAttributeRefactoring_RENAMED>
checkForAttributeMovesBetweenRemovedAndAddedClasses(
){^M
    List<UMLAttribute> addedAttributes = new
ArrayList<UMLAttribute>();
    for(UMLClass addedClass : addedClasses) {

        addedAttributes.addAll(addedClass.getAttributes());
@@ -586,13 +586,13 @@ public class UMLModelDiff {
        return checkForAttributeMoves(addedAttributes,
removedAttributes);
    }

- private List<MoveAttributeRefactoring>
checkForAttributeMoves(List<UMLAttribute> addedAttributes,
List<UMLAttribute> removedAttributes) {
-     List<MoveAttributeRefactoring> refactorings = new
ArrayList<MoveAttributeRefactoring>();
+ private List<MoveAttributeRefactoring_RENAMED>
checkForAttributeMoves(List<UMLAttribute> addedAttributes,
List<UMLAttribute> removedAttributes) {^M
+     List<MoveAttributeRefactoring_RENAMED>
refactorings = new
ArrayList<MoveAttributeRefactoring_RENAMED>();^M
        if(addedAttributes.size() <= removedAttributes.size()) {

```

```

        for(UMLAttribute addedAttribute : addedAttributes)
{
-
    List<MoveAttributeRefactoring> candidates =
new ArrayList<MoveAttributeRefactoring>();
+
    List<MoveAttributeRefactoring_RENAMED>
candidates = new
ArrayList<MoveAttributeRefactoring_RENAMED>();^M
        for(UMLAttribute removedAttribute :
removedAttributes) {
-
            MoveAttributeRefactoring candidate =
processPairOfAttributes(addedAttribute, removedAttribute);
+
            MoveAttributeRefactoring_RENAMED
candidate = processPairOfAttributes(addedAttribute,
removedAttribute);^M
            if(candidate != null) {
                candidates.add(candidate);
            }
@@ -602,9 +602,9 @@ public class UMLModelDiff {
}
else {
        for(UMLAttribute removedAttribute :
removedAttributes) {
-
            List<MoveAttributeRefactoring> candidates =
new ArrayList<MoveAttributeRefactoring>();
+
            List<MoveAttributeRefactoring_RENAMED>
candidates = new
ArrayList<MoveAttributeRefactoring_RENAMED>();^M
        for(UMLAttribute addedAttribute :
addedAttributes) {
-
            MoveAttributeRefactoring candidate =
processPairOfAttributes(addedAttribute, removedAttribute);
+
            MoveAttributeRefactoring_RENAMED
candidate = processPairOfAttributes(addedAttribute,
removedAttribute);^M
            if(candidate != null) {
                candidates.add(candidate);

```

```

    }

@@ -615,21 +615,21 @@ public class UMLModelDiff {
    return filterOutDuplicateRefactorings(refactorings);
}

- private List<MoveAttributeRefactoring>
filterOutDuplicateRefactorings(List<MoveAttributeRefactoring> refactorings) {
-     List<MoveAttributeRefactoring> filtered = new
ArrayList<MoveAttributeRefactoring>();
-     Map<String, List<MoveAttributeRefactoring>> map =
new LinkedHashMap<String, List<MoveAttributeRefactoring>>();
-     for(MoveAttributeRefactoring ref : refactorings) {
+     private List<MoveAttributeRefactoring_RENAMED>
filterOutDuplicateRefactorings(List<MoveAttributeRefactoring_RENAMED> refactorings) {^M
+     List<MoveAttributeRefactoring_RENAMED> filtered =
new ArrayList<MoveAttributeRefactoring_RENAMED>();^M
+     Map<String,
List<MoveAttributeRefactoring_RENAMED>> map = new
LinkedHashMap<String,
List<MoveAttributeRefactoring_RENAMED>>();^M
+     for(MoveAttributeRefactoring_RENAMED ref :
refactorings) {^M
         if(map.containsKey(ref.toString())) {
             map.get(ref.toString()).add(ref);
         }
         else {
-             List<MoveAttributeRefactoring> refs = new
ArrayList<MoveAttributeRefactoring>();
+             List<MoveAttributeRefactoring_RENAMED>
refs = new ArrayList<MoveAttributeRefactoring_RENAMED>();
()^M
             refs.add(ref);
             map.put(ref.toString(), refs);

```

```

        }

    }

    for(String key : map.keySet()) {
        -           List<MoveAttributeRefactoring> refs =
map.get(key);
        +           List<MoveAttributeRefactoring_RENAMED> refs =
map.get(key);^M
            if(refs.size() == 1) {
                filtered.addAll(refs);
            }
    }

@@ -640,24 +640,24 @@ public class UMLModelDiff {
    return filtered;
}

- private List<MoveAttributeRefactoring>
filterOutBasedOnFilePath(List<MoveAttributeRefactoring>
refs) {
    -   List<MoveAttributeRefactoring> filtered = new
ArrayList<MoveAttributeRefactoring>();
    -   Map<String, List<MoveAttributeRefactoring>>
groupBySourceFilePath = new LinkedHashMap<String,
List<MoveAttributeRefactoring>>();
    -   for(MoveAttributeRefactoring ref : refs) {
        + private List<MoveAttributeRefactoring_RENAMED>
filterOutBasedOnFilePath(List<MoveAttributeRefactoring_REN
AMED> refs) {^M
        +   List<MoveAttributeRefactoring_RENAMED> filtered =
new ArrayList<MoveAttributeRefactoring_RENAMED>();^M
        +   Map<String,
List<MoveAttributeRefactoring_RENAMED>>
groupBySourceFilePath = new LinkedHashMap<String,
List<MoveAttributeRefactoring_RENAMED>>();^M
        +   for(MoveAttributeRefactoring_RENAMED ref : refs) {^M
            String sourceFilePath =
ref.getOriginalAttribute().getLocationInfo().getFilePath();

```

```
if(groupBySourceFilePath.containsKey(sourceFilePath)) {  
  
    groupBySourceFilePath.get(sourceFilePath).add(ref);  
}  
  
else {  
-        List<MoveAttributeRefactoring> refs2 = new  
ArrayList<MoveAttributeRefactoring>();  
+        List<MoveAttributeRefactoring_RENAMED>  
refs2 = new ArrayList<MoveAttributeRefactoring_RENAMED>  
();^M  
        refs2.add(ref);  
        groupBySourceFilePath.put(sourceFilePath,  
refs2);  
}  
}  
  
for(String sourceFilePath :  
groupBySourceFilePath.keySet()) {  
-        List<MoveAttributeRefactoring>  
sourceFilePathGroup =  
groupBySourceFilePath.get(sourceFilePath);  
-        TreeMap<Integer, List<MoveAttributeRefactoring>>  
groupByLongestCommonSourceFi  
lePath = new TreeMap<Integer,  
List<MoveAttributeRefactoring>>();  
-        for(MoveAttributeRefactoring ref :  
sourceFilePathGroup) {  
+        List<MoveAttributeRefactoring_RENAMED>  
sourceFilePathGroup = groupBySourceFil  
ePath.get(sourceFilePath);^M  
+        TreeMap<Integer,  
List<MoveAttributeRefactoring_RENAMED>>  
groupByLongestCommonSourceFilePath = new  
TreeMap<Integer, List<MoveAttributeRefactoring_RENAMED>>  
();^M
```

```

+           for(MoveAttributeRefactoring_RENAMED ref :
sourceFilePathGroup) {^M
String longestCommonFilePathPrefix =
PrefixSuffixUtils.longestCommonPrefix(ref.getOriginalAttribut
e().getLocationInfo().getFilePath(),
ref.getMovedAttribute().getLocationInfo().getFilePath());
int length =
longestCommonFilePathPrefix.length();
@@ -665,7 +665,7 @@ public class UMLModelDiff {

groupByLongestCommonSourceFilePath.get(length).add(ref)
;
}
else {
-
List<MoveAttributeRefactoring> refs2 =
new ArrayList<MoveAttributeRefactoring>();
+
List<MoveAttributeRefactoring_RENAMED> refs2 = new
ArrayList<MoveAttributeRefactoring_RENAMED>();^M
refs2.add(ref);

groupByLongestCommonSourceFilePath.put(length, refs2);

}

@@ -675,16 +675,16 @@ public class UMLModelDiff {
return filtered;
}

-
private void
processCandidates(List<MoveAttributeRefactoring>
candidates, List<MoveAttributeRefactoring> refactorings) {
+
private void
processCandidates(List<MoveAttributeRefactoring_RENAMED
> candidates, List<MoveAttributeRefactoring_RENAMED>
refactorings) {^M

```

```

if(candidates.size() > 1) {
    -           TreeMap<Integer, List<MoveAttributeRefactoring>>
    map = new TreeMap<Integer, L
    ist<MoveAttributeRefactoring>>();
    -           for(MoveAttributeRefactoring candidate :
    candidates) {
        +           TreeMap<Integer,
    List<MoveAttributeRefactoring_RENAMED>> map = new
    TreeMap<Integer, List<MoveAttributeRefactoring_RENAMED>>
    ();^M
        +           for(MoveAttributeRefactoring_RENAMED
    candidate : candidates) {^M
            int compatibility =
    computeCompatibility(candidate);
            if(map.containsKey(compatibility)) {
                map.get(compatibility).add(candidate);
            }
            else {
                -           List<MoveAttributeRefactoring> refs =
    new ArrayList<MoveAttributeRefactoring>();
                +
    List<MoveAttributeRefactoring_RENAMED> refs = new
    ArrayList<MoveAttributeRefactoring_RENAMED>();^M
                refs.add(candidate);
                map.put(compatibility, refs);
            }
        }
    @@ -697,7 +697,7 @@ public class UMLModelDiff {
        }
    }
}

- private MoveAttributeRefactoring
processPairOfAttributes(UMLAttribute addedAttribute,
UMLAttribute removedAttribute) {
+ private MoveAttributeRefactoring_RENAMED
processPairOfAttributes(UMLAttribute addedAttribute,
UMLAttribute removedAttribute) {^M

```

```
if(addedAttribute.getName().equals(removedAttribute.getName()) &&

    addedAttribute.getType().equals(removedAttribute.getType())
)) {

    if(isSubclassOf(removedAttribute.getClassName(),
addedAttribute.getClassName())) {

@@ -711,7 +711,7 @@ public class UMLModelDiff {

        else

if(sourceClassImportsTargetClass(removedAttribute.getClassName(),
addedAttribute.getClassName()) ||

targetClassImportsSourceClass(removedAttribute.getClassName()
, addedAttribute.getClassName())) {

    if(!initializerContainsTypeLiteral(addedAttribute,
removedAttribute)) {

-           MoveAttributeRefactoring moveAttribute
= new MoveAttributeRefactoring(removedAttribute,
addedAttribute);

+           MoveAttributeRefactoring_RENAMED
moveAttribute = new
MoveAttributeRefactoring_RENAMED(removedAttribute,
addedAttribute);^M

           return moveAttribute;
    }
}

@@ -735,7 +735,7 @@ public class UMLModelDiff {

    return false;
}

- private int computeCompatibility(MoveAttributeRefactoring
candidate) {
```

```

+ private int
computeCompatibility(MoveAttributeRefactoring_RENAMED
candidate) {^M
    int count = 0;
    for(Refactoring ref : refactorings) {
        if(ref instanceof MoveOperationRefactoring) {
@@ -1113,7 +1113,7 @@ public class UMLModelDiff {
            ref = new
            PullUpAttributeRefactoring(removedAttribute,
addedAttribute);
        }
        else
if(parentType.equals(RefactoringType.EXTRACT_CLASS)) {
-
            ref = new
            MoveAttributeRefactoring(removedAttribute, addedAttribute);
+
            ref = new
            MoveAttributeRefactoring_RENAMED(removedAttribute,
addedAttribute);^M
        }
        else
if(parentType.equals(RefactoringType.EXTRACT_SUBCLASS))
{
            ref = new
PushDownAttributeRefactoring(removedAttribute,
addedAttribute);

```

Your task is to identify which refactoring type(s) have been applied in transforming the original program into the new version. Use only the following list of predefined refactorings:

(Add Method Parameter) – Introduces a new parameter to an existing method.

(Remove Method Parameter) – Eliminates an existing parameter from a method signature.

(Rename Method) – Changes the name of a method while preserving its behavior.

(Rename Class) – Changes the name of a class without altering its structure.

(Rename Package) – Changes the name of a package declaration.

(Rename Field) – Changes the name of a class or instance variable.

(Extract Class) – Moves a group of related fields and methods from an existing class into a newly created class.

(Extract Superclass) – Creates a new superclass to encapsulate shared attributes and behavior from two or more existing classes.

(Inline Method) – Replaces a method call with the method's body.

(Pull Up Method) – Pull a method from a child class (subclass) in the original code to its parent class (superclass) in the transformed version.

(Push Down Method) – Push a method from a parent class (superclass) in the original code to one or more of its child classes (subclasses) in the transformed version.

(Pull Up Field) – Pull a field from a child class (subclass) in the original code to its parent class (superclass) in the transformed version.

(Push Down Field) – Push a field from a parent class (superclass) in the original code to one or more of its child classes (subclasses) in the transformed version.

(Inline Class) – Merges a class into another when it is too small or redundant.

(Extract Interface) – Creates a new interface from existing method(s) in a class.

(Move Method) – Relocates a method to a more appropriate class.

(Move Field) – Relocates a field to a more appropriate class.

(Replace Magic Number with Constant) – Replaces a literal number with a named constant.

(Encapsulate Field) – Change a field access from not private to private and create get and set methods.

**\*\*Instructions:\*\***

1. Begin your response with a bullet-point list of the refactoring type(s) applied.
2. Then, briefly justify each identified refactoring with reference to the specific code changes.
3. Only include refactorings from the list above.
4. Be concise but precise in your explanations.

Do not generate explanations unrelated to the given transformation.

⋮

You are an expert coding assistant specialized in software refactoring, with many years of experience analyzing code transformations.

You will be given the diffs of a commit:

- **\*\*Diffs:\*\***

diff --git

src/gr/uom/java/xmi/decomposition/UMLOperationBodyMapper.java

src/gr/uom/java/xmi/decomposition/UMLOperationBodyMapper.java

index 958f34217..52539457d 100644

---

src/gr/uom/java/xmi/decomposition/UMLOperationBodyMapper.java

+++

src/gr/uom/java/xmi/decomposition/UMLOperationBodyMapper.java

```

@@ -154,7 +154,7 @@ public class
UMLOperationBodyMapper implements
Comparable<UMLOperationBodyMapper>
for(StatementObject statement :
getNonMappedLeavesT2()) {
    temporaryVariableAssignment(statement,
nonMappedLeavesT2);
}
-
for(StatementObject statement :
getNonMappedLeavesT1()) {
+
for(StatementObject statement :
getNonMappedLeavesT1_RENAMED()) {^M
    inlinedVariableAssignment(statement,
nonMappedLeavesT2);
}
}

@@ -198,7 +198,7 @@ public class UMLOperationBodyMapper
implements Comparable<UMLOperationBodyMapper>
for(StatementObject statement :
getNonMappedLeavesT2()) {
    temporaryVariableAssignment(statement,
nonMappedLeavesT2);
}
-
for(StatementObject statement :
getNonMappedLeavesT1()) {
+
for(StatementObject statement :
getNonMappedLeavesT1_RENAMED()) {^M
    inlinedVariableAssignment(statement,
nonMappedLeavesT2);
}
}

@@ -285,12 +285,12 @@ public class
UMLOperationBodyMapper implements
Comparable<UMLOperationBodyMapper>
OperationBody addedOperationBody =
addedOperation.getBody();

```

```

        if(addedOperationBody != null) {
            CompositeStatementObject composite2 =
addedOperationBody.getCompositeStatement();
            -           List<StatementObject> leaves1 =
operationBodyMapper.getNonMappedLeavesT1();
            +           List<StatementObject> leaves1 =
operationBodyMapper.getNonMappedLeavesT1_RENAMED();^
M
            List<CompositeStatementObject> innerNodes1 =
operationBodyMapper.getNonMappedInnerNodesT1();
            //adding leaves that were mapped with
replacements
            Set<StatementObject> addedLeaves1 = new
LinkedHashSet<StatementObject>();
            Set<CompositeStatementObject>
addedInnerNodes1 = new
LinkedHashSet<CompositeStatementObject>();
            -           for(StatementObject nonMappedLeaf1 : new
ArrayList<>
(operationBodyMapper.getNonMappedLeavesT1())) {
            +           for(StatementObject nonMappedLeaf1 : new
ArrayList<>
(operationBodyMapper.getNonMappedLeavesT1_RENAMED()))
) {^M

expandAnonymousAndLambdas(nonMappedLeaf1, leaves1,
innerNodes1,
addedLeaves1, addedInnerNodes1, operationBodyMapper);
        }
        for(AbstractCodeMapping mapping :
operationBodyMapper.getMappings()) {
@@ -428,7 +428,7 @@ public class
UMLOperationBodyMapper implements
Comparable<UMLOperationBodyMapper
        for(StatementObject statement :
getNonMappedLeavesT2()) {

```

```
        temporaryVariableAssignment(statement,
nonMappedLeavesT2);
    }
-           for(StatementObject statement :
getNonMappedLeavesT1()) {
+           for(StatementObject statement :
getNonMappedLeavesT1_RENAMED()) {^M
                inlinedVariableAssignment(statement,
nonMappedLeavesT2);
            }
        }
@@ -595,7 +595,7 @@ public class
UMLOperationBodyMapper implements
Comparable<UMLOperationBodyMapper
        for(StatementObject statement :
getNonMappedLeavesT2()) {
            temporaryVariableAssignment(statement,
nonMappedLeavesT2);
        }
-           for(StatementObject statement :
getNonMappedLeavesT1()) {
+           for(StatementObject statement :
getNonMappedLeavesT1_RENAMED()) {^M
                inlinedVariableAssignment(statement,
nonMappedLeavesT2);
            }
        }
@@ -642,7 +642,7 @@ public class
UMLOperationBodyMapper implements
Comparable<UMLOperationBodyMapper
        return mappings;
    }

-   public List<StatementObject> getNonMappedLeavesT1()
{
```

```

+    public List<StatementObject>
getNonMappedLeavesT1_RENAMED() {^M
    return nonMappedLeavesT1;
}

@@ -674,7 +674,7 @@ public class
UMLOperationBodyMapper implements
Comparable<UMLOperationBodyMapper
    nonMappedInnerNodeCount++;
}

int nonMappedLeafCount = 0;
-
for(StatementObject statement :
getNonMappedLeavesT1()) {
+
for(StatementObject statement :
getNonMappedLeavesT1_RENAMED()) {^M
    if(statement.countableStatement())
        nonMappedLeafCount++;
}

@@ -683,7 +683,7 @@ public class
UMLOperationBodyMapper implements
Comparable<UMLOperationBodyMapper

public int nonMappedLeafElementsT1() {
    int nonMappedLeafCount = 0;
-
for(StatementObject statement :
getNonMappedLeavesT1()) {
+
for(StatementObject statement :
getNonMappedLeavesT1_RENAMED()) {^M
    if(statement.countableStatement())
        nonMappedLeafCount++;
}

@@ -792,7 +792,7 @@ public class
UMLOperationBodyMapper implements
Comparable<UMLOperationBodyMapper
}

}

```

```

        int nonMappedLeafCount = 0;
-
-         for(StatementObject statement :
getNonMappedLeavesT1()) {
+
+         for(StatementObject statement :
getNonMappedLeavesT1_RENAMED()) {^M
+
+             if(statement.countableStatement()) {
+
+                 Map<String, List<OperationInvocation>>
methodInvocationMap =
statement.getMethodInvocationMap();
+
+                 for(String key :
methodInvocationMap.keySet()) {
+
@@ -3948,7 +3948,7 @@ public class
UMLOperationBodyMapper implements
Comparable<UMLOperationBodyMapper
}

public boolean isEmpty() {
-
-             return getNonMappedLeavesT1().isEmpty() &&
getNonMappedInnerNodesT1().isEmpty()
&&
+
+             return
getNonMappedLeavesT1_RENAMED().isEmpty() &&
getNonMappedInnerNodesT1().isEmpty() &&^M
+
+                 getNonMappedLeavesT2().isEmpty() &&
getNonMappedInnerNodesT2().i
sEmpty();
}

diff --git
src/gr/uom/java/xmi/decomposition/VariableReplacementAnaly
sis.java
src/gr/uom/java/xmi/decomposition/VariableReplacementAnaly
sis.java
index 31c83ffc1..a9cfe4b3f 100644
---
src/gr/uom/java/xmi/decomposition/VariableReplacementAnaly

```

```
sis.java

+++
src/gr/uom/java/xmi/decomposition/VariableReplacementAnaly
sis.java
@@ -60,7 +60,7 @@ public class
VariableReplacementAnalysis {

    public
VariableReplacementAnalysis(UMLOperationBodyMapper
mapper, Set<Refactoring> refac
torings, UMLClassBaseDiff classDiff) {
        this.mappings = mapper.getMappings();
-
        this.nonMappedLeavesT1 =
mapper.getNonMappedLeavesT1();
+
        this.nonMappedLeavesT1 =
mapper.getNonMappedLeavesT1_RENAMED();
        this.nonMappedLeavesT2 =
mapper.getNonMappedLeavesT2();
        this.nonMappedInnerNodesT1 =
mapper.getNonMappedInnerNodesT1();
        this.nonMappedInnerNodesT2 =
mapper.getNonMappedInnerNodesT2();
diff --git
src/gr/uom/java/xmi/diff/ExtractOperationDetection.java
src/gr/uom/java/xmi/diff/ExtractOperationDetection.java
index 47945d29e..ec6dbc1fe 100644
--- src/gr/uom/java/xmi/diff/ExtractOperationDetection.java
+++ src/gr/uom/java/xmi/diff/ExtractOperationDetection.java
@@ -38,7 +38,7 @@ public class ExtractOperationDetection {

    public List<ExtractOperationRefactoring>
check(UMLOperation addedOperation) throws Refac
toringMinerTimedOutException {
        List<ExtractOperationRefactoring> refactorings =
new ArrayList<ExtractOperationRefactoring>();
```

```

-           if(!mapper.getNonMappedLeavesT1().isEmpty() ||
!mapper.getNonMappedInnerNodesT1().isEmpty() ||
+
if(!mapper.getNonMappedLeavesT1_RENAMED().isEmpty() ||
!mapper.getNonMappedInnerNodesT1().isEmpty() ||

!mapper.getReplacementsInvolvingMethodInvocation().isEmpt
y()) {

    List<OperationInvocation>
addedOperationInvocations = matchingInvocation
s(addedOperation, operationInvocations,
mapper.getOperation2().variableTypeMap());

    if(addedOperationInvocations.size() > 0) {
@@ -99,8 +99,8 @@ public class ExtractOperationDetection {
                if(!mapping.isExact() ||
mapping.getFragment1().getString().equals("{}") {

                    AbstractCodeFragment
fragment1 = mapping.getFragment1();

                    if(fragment1 instanceof
StatementObject) {

-
if(!mapper.getNonMappedLeavesT1().contains(fragment1)) {

-
mapper.getNonMappedLeavesT1().add((StatementObject)fra
gment1);

+
if(!mapper.getNonMappedLeavesT1_RENAMED().contains(fra
gment1)) {

+
mapper.getNonMap
pedLeavesT1_RENAMED().add((StatementObject)fragment1);
}

}
else if(fragment1
instanceof CompositeStatementObject) {

```

```
diff --git
src/gr/uom/java/xmi/diff/InlineOperationDetection.java
src/gr/uom/java/xmi/diff/InlineOperationDetection.java
index 72c56ab85..fc6e057f0 100644
--- src/gr/uom/java/xmi/diff/InlineOperationDetection.java
+++ src/gr/uom/java/xmi/diff/InlineOperationDetection.java
@@ -114,7 +114,7 @@ public class InlineOperationDetection {

    private List<OperationInvocation>
getInvocationsInTargetOperationBeforeInline(UMLOperationB
odyMapper mapper) {
    List<OperationInvocation> operationInvocations =
mapper.getOperation1().getAllOp
erationInvocations();
-    for(StatementObject statement :
mapper.getNonMappedLeavesT1()) {
+    for(StatementObject statement :
mapper.getNonMappedLeavesT1_RENAMED()) {
        ExtractOperationDetection.addStatementInvocations(operation
nInvocations,
statement);
        for(UMLAnonymousClass anonymousClass :
classDiff.getRemovedAnonymousClasses()) {
if(statement.getLocationInfo().subsumes(anonymousClass.ge
tLocationInfo())) {
@@ -133,7 +133,7 @@ public class InlineOperationDetection {

    private boolean
inlineMatchCondition(UMLOperationBodyMapper
operationBodyMapper) {
        int delegateStatements = 0;
-        for(StatementObject statement :
operationBodyMapper.getNonMappedLeavesT1()) {
```

```

+           for(StatementObject statement :
operationBodyMapper.getNonMappedLeavesT1_RENAMED())
{

```

OperationInvocation invocation =  
statement.invocationCoveringEntireFragment();  
if(invocation != null &&  
invocation.matchesOperation(operationBodyMapper.getOper  
ation1())) {  
 delegateStatements++;  
diff --git  
src/gr/uom/java/xmi/diff/InlineOperationRefactoring.java  
src/gr/uom/java/xmi/diff/InlineOperationRefactoring.java  
index eb9ea5fed..c479d7f10 100644  
--- src/gr/uom/java/xmi/diff/InlineOperationRefactoring.java  
+++ src/gr/uom/java/xmi/diff/InlineOperationRefactoring.java  
@@ -184,7 +184,7 @@ public class InlineOperationRefactoring  
implements Refactoring {  
 setDescription("inlined method  
invocation")  
  
.setCodeElement(invocation.actualString()));  
}  
- for(StatementObject statement :  
bodyMapper.getNonMappedLeavesT1()) {  
+ for(StatementObject statement :  
bodyMapper.getNonMappedLeavesT1\_RENAMED()) {^M  
 ranges.add(statement.codeRange()).  
 setDescription("deleted statement in  
inlined method declaration"));  
}  
diff --git src/gr/uom/java/xmi/diff/UMLClassBaseDiff.java  
src/gr/uom/java/xmi/diff/UMLClassBaseDiff.java  
index 6ee124277..e78f18e0b 100644  
--- src/gr/uom/java/xmi/diff/UMLClassBaseDiff.java  
+++ src/gr/uom/java/xmi/diff/UMLClassBaseDiff.java

```

@@ -709,7 +709,7 @@ public abstract class
UMLClassBaseDiff implements
Comparable<UMLClassBaseDiff> {
    matchingVariableName =
variableDeclaration.getVariableName();
}
else {
-
    for(StatementObject
statement :
candidateMapper.getNonMappedLeavesT1()) {
+
    for(StatementObject
statement :
candidateMapper.getNonMappedLeavesT1_RENAMED()) {

if(statement.getString().startsWith(variableDeclaration.getVar
iableName() + "=") ||
statement.getString().startsWith("this." +
variableDeclaration.getVariableName() + "=")) {

nonMatchingVariableNames.add(variableDeclaration.getVaria
bleName());
@@ -1162,7 +1162,7 @@ public abstract class
UMLClassBaseDiff implements
Comparable<UMLClassBaseDiff> {
    int
parameterizedVariableDeclarationStatements = 0;
    UMLOperation addedOperation =
operationBodyMapper.getOperation2();
    List<String> nonMappedLeavesT1 = new
ArrayList<String>();
-
    for(StatementObject statement :
operationBodyMapper.getNonMappedLeavesT1()) {
+
    for(StatementObject statement :
operationBodyMapper.getNonMappedLeavesT1_RENAMED())
{

```

```

        if(statement.countableStatement()) {

            nonMappedLeavesT1.add(statement.getString());
                for(String parameterName :
                    addedOperation.getParameterNameList()) {
                        @@ -1207,7 +1207,7 @@ public abstract class
                        UMLClassBaseDiff implements
                        Comparable<UMLClassBaseDiff> {

                operationBodyMapper.getNonMappedInnerNodesT1().size()
                == 0 &&
                operationBodyMapper.getNonMappedInnerNodesT2().size()
                == 0) {

                    StatementObject
                    statementUsingParameterAsInvoker1 = null;
                    UMOOperation removedOperation =
                    operationBodyMapper.getOperation1();
                    -
                        for(StatementObject statement :
                            operationBodyMapper.getNonMappedLeavesT1()) {
                    +
                        for(StatementObject statement :
                            operationBodyMapper.getNonMapped
                            LeavesT1_RENAMED()) {
                            if(statement.countableStatement()) {
                                for(String parameterName :
                                    removedOperation.getParameterNameList()) {
  OperationInvocation
  invocation = statement.invocationCoveringEntireFragment();
  @@ -1359,7 +1359,7 @@ public abstract class
  UMLClassBaseDiff implements
  Comparable<UMLClassBaseDiff> {

                            }

                    private boolean
                    singleUnmatchedStatementCallsAddedOperation(UMLOperati
                    onBodyMapper operationBodyMapper) {

```

```

-           List<StatementObject> nonMappedLeavesT1 =
operationBodyMapper.getNonMappedLeavesT1();

+           List<StatementObject> nonMappedLeavesT1 =
operationBodyMapper.getNonMappedLeaves
T1_RENAMED();

           List<StatementObject> nonMappedLeavesT2 =
operationBodyMapper.getNonMappedLeavesT2();

           if(nonMappedLeavesT1.size() == 1 &&
nonMappedLeavesT2.size() == 1) {

               StatementObject statementT2 =
nonMappedLeavesT2.get(0);

diff --git src/gr/uom/java/xmi/diff/UMLModelDiff.java
src/gr/uom/java/xmi/diff/UMLModelDiff.javaindex
c975c153c..9a2913e71 100644
--- src/gr/uom/java/xmi/diff/UMLModelDiff.java
+++ src/gr/uom/java/xmi/diff/UMLModelDiff.java
@@ -1776,7 +1776,7 @@ public class UMLModelDiff {

}

int delegateStatements = 0;

-           for(StatementObject statement :
operationBodyMapper.getNonMappedLeavesT1()) {
+           for(StatementObject statement :
operationBodyMapper.getNonMappedLeavesT1_RENAMED())
{^M

               OperationInvocation invocation =
statement.invocationCoveringEntireFragment();

               if(invocation != null &&
invocation.matchesOperation(operationBodyMapper.getOper
ation1())) {

                   delegateStatements++;
@@ -2307,7 +2307,7 @@ public class UMLModelDiff {

               int nonMappedElementsT2 =
operationBodyMapper.nonMappedElementsT2();

               UMLClass addedClass =
getAddedClass(operationBodyMapper.getOperation2().getCl

```

```
ssName());  
        int nonMappedStatementsDeclaringSameVariable =  
0;  
-         for(ListIterator<StatementObject> leafIterator1 =  
operationBodyMapper.getNonMappedLeavesT1().listIterator();  
leafIterator1.hasNext();) {  
+         for(ListIterator<StatementObject> leafIterator1 =  
operationBodyMapper.getNonMappedLeavesT1_RENAMED().li  
stIterator(); leafIterator1.hasNext();) {^M  
            StatementObject s1 = leafIterator1.next();  
            for(StatementObject s2 :  
operationBodyMapper.getNonMappedLeavesT2()) {  
                if(s1.getVariableDeclarations().size() == 1 &&  
s2.getVariableDeclarations().size() == 1) {
```

Your task is to identify which refactoring type(s) have been applied in transforming the original program into the new version. Use only the following list of predefined refactorings:

(Add Method Parameter) – Introduces a new parameter to an existing method.

(Remove Method Parameter) – Eliminates an existing parameter from a method signature.

(Rename Method) – Changes the name of a method while preserving its behavior.

(Rename Class) – Changes the name of a class without altering its structure.

(Rename Package) – Changes the name of a package declaration.

(Rename Field) – Changes the name of a class or instance variable.

(Extract Class) – Moves a group of related fields and methods from an existing class into a newly created class.

(Extract Superclass) – Creates a new superclass to encapsulate shared attributes and behavior from two or more existing classes.

(Inline Method) – Replaces a method call with the method's body.

(Pull Up Method) – Pull a method from a child class (subclass) in the original code to its parent class (superclass) in the transformed version.

(Push Down Method) – Push a method from a parent class (superclass) in the original code to one or more of its child classes (subclasses) in the transformed version.

(Pull Up Field) – Pull a field from a child class (subclass) in the original code to its parent class (superclass) in the transformed version.

(Push Down Field) – Push a field from a parent class (superclass) in the original code to one or more of its child classes (subclasses) in the transformed version.

(Inline Class) – Merges a class into another when it is too small or redundant.

(Extract Interface) – Creates a new interface from existing method(s) in a class.

(Move Method) – Relocates a method to a more appropriate class.

(Move Field) – Relocates a field to a more appropriate class.

(Replace Magic Number with Constant) – Replaces a literal number with a named constant.

(Encapsulate Field) – Change a field access from not private to private and create get and set methods.

**\*\*Instructions:\*\***

1. Begin your response with a bullet-point list of the refactoring type(s) applied.
2. Then, briefly justify each identified refactoring with reference to the specific code changes.
3. Only include refactorings from the list above.
4. Be concise but precise in your explanations.

Do not generate explanations unrelated to the given transformation.

You are an expert coding assistant specialized in software refactoring, with many years of experience analyzing code transformations.

You will be given the diffs of a commit:

- **Diff:**

```
diff --git src-
test/org/refactoringminer/utils/RefactoringSet.java src-
test/org/refactoringminer/utils/RefactoringSet.java
index 72cd2d480..41302b7fa 100644
--- src-test/org/refactoringminer/utils/RefactoringSet.java
+++ src-test/org/refactoringminer/utils/RefactoringSet.java
@@ -35,7 +35,7 @@ public class RefactoringSet {
    return project;
}
```

- public String getRevision() {

+ public String getRevision\_RENAMED() {^M
 return revision;
}

```
diff --git src-
test/org/refactoringminer/utils/ResultComparator.java src-
test/org/refactoringminer/utils/ResultComparator.java
index 057b1b631..4af0154c7 100644
--- src-test/org/refactoringminer/utils/ResultComparator.java
+++ src-test/org/refactoringminer/utils/ResultComparator.java
@@ -45,14 +45,14 @@ public class ResultComparator {
```

```
    public ResultComparator expect(RefactoringSet ... sets) {
```

```

        for (RefactoringSet set : sets) {
-
            expectedMap.put(getProjectRevisionId(set.getProject(),
set.getRevision()), set);
+
            expectedMap.put(getProjectRevisionId(set.getProject(),
set.getRevision_RENAMED()),
set);^M
        }
        return this;
    }

    public ResultComparator dontExpect(RefactoringSet ...
sets) {
        for (RefactoringSet set : sets) {
-
            notExpectedMap.put(getProjectRevisionId(set.getProject(),
set.getRevision()), set);+
            notExpectedMap.put(getProjectRevisionId(set.getProject(),
set.getRevision_RENAMED()), set);^M
        }
        return this;
    }

@@ -60,7 +60,7 @@ public class ResultComparator {
    public ResultComparator compareWith(String groupId,
RefactoringSet ... actualArray) {
        for (RefactoringSet actual : actualArray) {
            groupIds.add(groupId);
-
            resultMap.put(getResultId(actual.getProject(),
actual.getRevision(), groupId), actual);
+
            resultMap.put(getResultId(actual.getProject(),
actual.getRevision_RENAMED(), groupId), actual);^M
        }
        return this;
    }

@@ -94,7 +94,7 @@ public class ResultComparator {

```

```

EnumSet<RefactoringType> ignore =
EnumSet.complementOf(refTypesToConsider);

for (RefactoringSet expected : expectedMap.values()) {
-   RefactoringSet actual =
resultMap.get(getResultId(expected.getProject(),
expected.getRevision(), groupId));
+   RefactoringSet actual =
resultMap.get(getResultId(expected.getProject(),
expected.getRevision_RENAMED(), groupId));^M
    if (actual != null) {
        Set<RefactoringRelationship> expectedRefactorings
=
expected.ignoring(ignore).ignoringMethodParameters(ignore
MethodParams).getRefactorings();
        Set<RefactoringRelationship> actualRefactorings =
actual.ignoring(ignore).ignoringMethodParameters(ignoreMet
hodParams).getRefactorings();
@@ -157,7 +157,7 @@ public class ResultComparator {
        for (String groupId : groupIds) {
            header.append('\t');
            header.append(groupId);
-            RefactoringSet actual =
resultMap.get(getResultId(expected.getProject(),
expected.getRevision(), groupId));
+            RefactoringSet actual =
resultMap.get(getResultId(expected.getProject(),
expected.getRevision_RENAMED(), groupId));^M
            if (actual != null) {
                all.addAll(actual.ignoring(ignore).ignoringMethodParameters(
ignoreMethodParams).getRefactorings()); //
}
@@ -167,14 +167,14 @@ public class ResultComparator {
            headerPrinted = true;
}

```

```

        if (!all.isEmpty()) {
-
        out.println(getProjectRevisionId(expected.getProject(),
expected.getRevision())
);
+
        out.println(getProjectRevisionId(expected.getProject(),
expected.getRevision_RENAMED()));^M
        ArrayList<RefactoringRelationship> allList = new
ArrayList<>();
        allList.addAll(all);
        Collections.sort(allList);
        for (RefactoringRelationship r : allList) {
            out.print(r.toString());
            for (String groupId : groupIds) {
-
                RefactoringSet actual =
resultMap.get(getResultId(expected.getProject(),
expected.getRevision(), groupId));
+
                RefactoringSet actual =
resultMap.get(getResultId(expected.getProject(),
expected.getRevision_RENAMED(), groupId));^M
                out.print('\t');
                if (actual != null) {
                    Set<RefactoringRelationship>
actualRefactorings =
actual.ignoring(ignore).ignoringMethodParameters(ignoreMet
hodParams).getRefactorings();
@@ -352,7 +352,7 @@ public class ResultComparator {
                    public static RefactoringSet[]
collectRmResult(GitHistoryRefactoringMiner rm,
RefactoringSet[] oracle) {
                    RefactoringSet[] result = new
RefactoringSet[oracle.length];
                    for (int i = 0; i < result.length; i++) {
-
                        result[i] = collectRmResult(rm, oracle[i].getProject(),
oracle[i].getRevision());

```

```
+      result[i] = collectRmResult(rm, oracle[i].getProject(),
oracle[i].getRevision_RENAMED());^M
}
return result;
}
```

Your task is to identify which refactoring type(s) have been applied in transforming the original program into the new version. Use only the following list of predefined refactorings:

(Add Method Parameter) – Introduces a new parameter to an existing method.

(Remove Method Parameter) – Eliminates an existing parameter from a method signature.

(Rename Method) – Changes the name of a method while preserving its behavior.

(Rename Class) – Changes the name of a class without altering its structure.

(Rename Package) – Changes the name of a package declaration.

(Rename Field) – Changes the name of a class or instance variable.

(Extract Class) – Moves a group of related fields and methods from an existing class into a newly created class.

(Extract Superclass) – Creates a new superclass to encapsulate shared attributes and behavior from two or more existing classes.

(Inline Method) – Replaces a method call with the method's body.

(Pull Up Method) – Pull a method from a child class (subclass) in the original code to its parent class (superclass) in the transformed version.

(Push Down Method) – Push a method from a parent class (superclass) in the original code to one or more of its child classes (subclasses) in the transformed version.

(Pull Up Field) – Pull a field from a child class (subclass) in the original code to its parent class (superclass) in the transformed version.

(Push Down Field) – Push a field from a parent class (superclass) in the original code to one or more of its child classes (subclasses) in the transformed version.

(Inline Class) – Merges a class into another when it is too small or redundant.

(Extract Interface) – Creates a new interface from existing method(s) in a class.

(Move Method) – Relocates a method to a more appropriate class.

(Move Field) – Relocates a field to a more appropriate class.

(Replace Magic Number with Constant) – Replaces a literal number with a named constant.

(Encapsulate Field) – Change a field access from not private to private and create get and set methods.

**\*\*Instructions:\*\***

1. Begin your response with a bullet-point list of the refactoring type(s) applied.
2. Then, briefly justify each identified refactoring with reference to the specific code changes.
3. Only include refactorings from the list above.
4. Be concise but precise in your explanations.

Do not generate explanations unrelated to the given transformation.

⋮

You are an expert coding assistant specialized in software refactoring, with many years of experience analyzing code transformations.

You will be given the diffs of a commit:

- **\*\*Diffs:\*\***

```
diff --git src/gr/uom/java/xmi/UMLAttribute.java
src/gr/uom/java/xmi/UMLAttribute.java
index bd6e82530..827846a6f 100644
--- src/gr/uom/java/xmi/UMLAttribute.java
+++ src/gr/uom/java/xmi/UMLAttribute.java
@@ -45,7 +45,7 @@ public class UMLAttribute implements
Comparable<UMLAttribute>, Serializable, Loc
        this.visibility = visibility;
    }
```

```
-    public boolean isFinal() {
+    public boolean isFinal_RENAMED() {^M
        return isFinal;
    }
```

```
diff --git src/gr/uom/java/xmi/diff/UMLAttributeDiff.java
src/gr/uom/java/xmi/diff/UMLAttributeDiff.java
index 0638c8f00..6b7f6fe71 100644
--- src/gr/uom/java/xmi/diff/UMLAttributeDiff.java
+++ src/gr/uom/java/xmi/diff/UMLAttributeDiff.java
@@ -42,7 +42,7 @@ public class UMLAttributeDiff {
        qualifiedTypeChanged = true;
        if(removedAttribute.isStatic() != addedAttribute.isStatic())
            staticChanged = true;
-        if(removedAttribute.isFinal() != addedAttribute.isFinal())
+        if(removedAttribute.isFinal_RENAMED() != addedAttribute.isFinal_RENAMED())^M
            finalChanged = true;
        this.annotationListDiff = new
UMLAnnotationListDiff(removedAttribute.getAnnotations()),
```

```

addedAttribute.getAnnotations());
}

@@ -89,8 +89,8 @@ public class UMLAttributeDiff {
    sb.append("\t").append("modifier changed from
" + (removedAttribute.isStatic() ? "static" : "non-static") + " to
" +
    (addedAttribute.isStatic() ? "static" :
"non-static")).append("\n");
    if(finalChanged)
        -
            sb.append("\t").append("modifier changed
from " + (removedAttribute.isFinal() ? "final" : "non-final") + " to " +
to " +
        -
            (addedAttribute.isFinal() ? "final" :
"non-final")).append("\n");
        +
            sb.append("\t").append("modifier changed
from " + (removedAttribute.isFinal_RENAMED() ? "final" :
"non-final") + " to " +^M
        +
            (addedAttribute.isFinal_RENAMED() ?
"final" : "non-final")).append("\n");^M
    for(UMLAnnotation annotation :
annotationListDiff.getRemovedAnnotations()) {
        sb.append("\t").append("annotation " +
annotation + " removed").append("\n");
    }
}

```

Your task is to identify which refactoring type(s) have been applied in transforming the original program into the new version. Use only the following list of predefined refactorings:

(Add Method Parameter) – Introduces a new parameter to an existing method.

(Remove Method Parameter) – Eliminates an existing parameter from a method signature.

(Rename Method) – Changes the name of a method while preserving its behavior.

(Rename Class) – Changes the name of a class without altering its structure.

(Rename Package) – Changes the name of a package declaration

# Samples – Lettuce

<https://g.co/gemini/share/ae31d722c70e>



Created with 2.5 Pro (preview) May 14, 2025 at 08:17 PM • Published May 16, 2025 at 07:47 PM

You are an expert coding assistant specialized in software refactoring, with many years of experience analyzing code transformations.

You will be given the diffs of a commit:

```
- **Diffs:**  
diff --git  
a/src/main/java/io/lettuce/core/dynamic/CommandSegmentCo  
mmandFactory.java  
b/src/main/java/io/lettuce/core/dynamic/CommandSegmentCo  
mmandFactory.java  
index 732b58b97..68c4fbea6 100644  
---  
a/src/main/java/io/lettuce/core/dynamic/CommandSegmentCo  
mmandFactory.java  
+++  
b/src/main/java/io/lettuce/core/dynamic/CommandSegmentCo  
mmandFactory.java  
@@ -29,7 +29,7 @@ class  
CommandSegmentCommandFactory implements  
CommandFactory {  
  
private final RedisCodec<Object, Object> redisCodec;
```

```
- private final ParameterBinder parameterBinder = new
ParameterBinder();
+ private final RedisParameterBinder redisParameterBinder
= new RedisParameterBinder();

    private final CommandOutputFactory outputFactory;

@@ -84,7 +84,7 @@ class
CommandSegmentCommandFactory implements
CommandFactory {
    CommandOutput<Object, Object, ?> output =
outputFactory.create(redisCodec);
    Command<Object, Object, ?> command = new
Command<>(this.segments.getCommandType(), output,
args);

-     parameterBinder.bind(args, redisCodec, segments,
parametersAccessor);
+     redisParameterBinder.bind(args, redisCodec, segments,
parametersAccessor);

        return (Command) command;
    }
diff --git
a/src/main/java/io/lettuce/core/dynamic/ParameterBinder.java
b/src/main/java/io/lettuce/core/dynamic/RedisParameterBinder
.java
similarity index 99%
rename from
src/main/java/io/lettuce/core/dynamic/ParameterBinder.java
rename to
src/main/java/io/lettuce/core/dynamic/RedisParameterBinder.ja
va
index 9669d29f2..fb889f440 100644
---
a/src/main/java/io/lettuce/core/dynamic/ParameterBinder.java
```

```
+++
b/src/main/java/io/lettuce/core/dynamic/RedisParameterBinder
.java
@@ -21,7 +21,7 @@ import
io.lettuce.core.protocol.ProtocolKeyword;
 * @author Mark Paluch
 * @since 5.0
*/
-class ParameterBinder {
+class RedisParameterBinder {

    private static final byte[] MINUS_BYTES = { '-' };

diff --git
a/src/test/java/io/lettuce/core/dynamic/ParameterBinderUnitTe
sts.java
b/src/test/java/io/lettuce/core/dynamic/RedisParameterBinder
UnitTests.java
similarity index 98%
rename from
src/test/java/io/lettuce/core/dynamic/ParameterBinderUnitTest
s.java
rename to
src/test/java/io/lettuce/core/dynamic/RedisParameterBinderUn
itTests.java
index 1808904da..d3b709495 100644
---
a/src/test/java/io/lettuce/core/dynamic/ParameterBinderUnitTe
sts.java
+++
b/src/test/java/io/lettuce/core/dynamic/RedisParameterBinder
UnitTests.java
@@ -25,9 +25,9 @@ import
io.lettuce.core.protocol.CommandType;
*/
@Tag(UNIT_TEST)
```

```
@ExtendWith(MockitoExtension.class)
-class ParameterBinderUnitTests {
+class RedisParameterBinderUnitTests {

    - private ParameterBinder binder = new ParameterBinder();
    + private RedisParameterBinder binder = new
        RedisParameterBinder();

    private CommandSegments segments = new
        CommandSegments(Collections.singletonList(CommandSegm
        ent.constant("set")));
}
```

Your task is to identify which refactoring type(s) have been applied in transforming the original program into the new version. Use only the following list of predefined refactorings:

(Add Method Parameter) – Introduces a new parameter to an existing method.

(Remove Method Parameter) – Eliminates an existing parameter from a method signature.

(Rename Method) – Changes the name of a method while preserving its behavior.

(Rename Class) – Changes the name of a class without altering its structure.

(Rename Package) – Changes the name of a package declaration.

(Rename Field) – Changes the name of a class or instance variable.

(Extract Class) – Moves a group of related fields and methods from an existing class into a newly created class.

(Extract Superclass) – Creates a new superclass to encapsulate shared attributes and behavior from two or more existing classes.

(Inline Method) – Replaces a method call with the method's body.

(Pull Up Method) – Pull a method from a child class (subclass) in the original code to its parent class (superclass) in the transformed version.

(Push Down Method) – Push a method from a parent class (superclass) in the original code to one or more of its child classes (subclasses) in the transformed version.

(Pull Up Field) – Pull a field from a child class (subclass) in the original code to its parent class (superclass) in the transformed version.

(Push Down Field) – Push a field from a parent class (superclass) in the original code to one or more of its child classes (subclasses) in the transformed version.

(Inline Class) – Merges a class into another when it is too small or redundant.

(Extract Interface) – Creates a new interface from existing method(s) in a class.

(Move Method) – Relocates a method to a more appropriate class.

(Move Field) – Relocates a field to a more appropriate class.

(Replace Magic Number with Constant) – Replaces a literal number with a named constant.

(Encapsulate Field) – Change a field access from not private to private and create get and set methods.

**\*\*Instructions:\*\***

1. Begin your response with a bullet-point list of the refactoring type(s) applied.
2. Then, briefly justify each identified refactoring with reference to the specific code changes.
3. Only include refactorings from the list above.
4. Be concise but precise in your explanations.

Do not generate explanations unrelated to the given transformation.



- Rename Class
- Rename Field

#### Justifications:

- **Rename Class:**

- The class `ParameterBinder` located in the file `src/main/java/io/lettuce/core/dynamic/ParameterBinder.java` was renamed to `RedisParameterBinder`. This is evident from the file rename operation (`rename from src/main/java/io/lettuce/core/dynamic/ParameterBinder.java rename to src/main/java/io/lettuce/core/dynamic/RedisParameterBinder.java`) and the change in the class declaration from `class ParameterBinder` to `class RedisParameterBinder`.
- Similarly, the test class `ParameterBinderUnitTests` in `src/test/java/io/lettuce/core/dynamic/ParameterBinderUnitTests.java` was renamed to `RedisParameterBinderUnitTests`. This is shown by the file rename (`rename from src/test/java/io/lettuce/core/dynamic/ParameterBinderUnitTests.java rename to src/test/java/io/lettuce/core/dynamic/RedisParameterBinderUnitTests.java`) and the class declaration change from `class ParameterBinderUnitTests` to `class RedisParameterBinderUnitTests`.

- **Rename Field:**

- In the class `CommandSegmentCommandFactory`, the instance field `parameterBinder` was renamed to `redisParameterBinder`. The declaration changed from `private final ParameterBinder parameterBinder = new ParameterBinder();` to `private final RedisParameterBinder redisParameterBinder = new RedisParameterBinder();`. Its usage was also updated accordingly from `parameterBinder.bind(...)` to `redisParameterBinder.bind(...)`.

You are an expert coding assistant specialized in software refactoring, with many years of experience analyzing code transformations.

You will be given the diffs of a commit:

```
- **Diffs:**  
diff --git  
a/src/main/java/io/lettuce/core/dynamic/RedisCommandFactor  
y.java  
b/src/main/java/io/lettuce/core/dynamic/RedisCommandFactor  
y.java  
index ae2111f18..c300cfccf2 100644  
----  
a/src/main/java/io/lettuce/core/dynamic/RedisCommandFactor  
y.java  
+++  
b/src/main/java/io/lettuce/core/dynamic/RedisCommandFactor  
y.java  
@@ -15,7 +15,7 @@ import  
io.lettuce.core.codec.ByteArrayCodec;  
import io.lettuce.core.codec.RedisCodec;  
import io.lettuce.core.codec.StringCodec;  
import io.lettuce.core.dynamic.batch.BatchSize;  
-import  
io.lettuce.core.dynamic.intercept.DefaultMethodInvokingInter  
ceptor;  
+import  
io.lettuce.core.dynamic.intercept.DefaultMethodInvoker;  
import  
io.lettuce.core.dynamic.intercept.InvocationProxyFactory;  
import io.lettuce.core.dynamic.intercept.MethodInterceptor;  
import io.lettuce.core.dynamic.intercept.MethodInvocation;  
@@ -181,7 +181,7 @@ public class RedisCommandFactory {  
    BatchAwareCommandLookupStrategy lookupStrategy =  
new BatchAwareCommandLookupStrategy(  
        new CompositeCommandLookupStrategy(),  
metadata);
```

```
-    factory.addInterceptor(new
DefaultMethodInvokingInterceptor());
+    factory.addInterceptor(new DefaultMethodInvoker());
    factory.addInterceptor(new
CommandFactoryExecutorMethodInterceptor(metadata,
lookupStrategy));

        return
factory.createProxy(commandInterface.getClassLoader());
diff --git
a/src/main/java/io/lettuce/core/dynamic/intercept/DefaultMeth
odInvokingInterceptor.java
b/src/main/java/io/lettuce/core/dynamic/intercept/DefaultMeth
odInvoker.java
similarity index 92%
rename from
src/main/java/io/lettuce/core/dynamic/intercept/DefaultMeth
odInvokingInterceptor.java
rename to
src/main/java/io/lettuce/core/dynamic/intercept/DefaultMeth
odInvoker.java
index c5fcc4ed2..624e047cc 100644
---
a/src/main/java/io/lettuce/core/dynamic/intercept/DefaultMeth
odInvokingInterceptor.java
+++
b/src/main/java/io/lettuce/core/dynamic/intercept/DefaultMeth
odInvoker.java
@@ -17,7 +17,7 @@ import
io.lettuce.core.internal.LettuceAssert;
 * @see MethodInvocation
 * @see InvocationTargetException
 */
-public class DefaultMethodInvokingInterceptor implements
MethodInterceptor {
```

```

+public class DefaultMethodInvoker implements
MethodInterceptor {

    private final Map<Method, MethodHandle>
methodHandleCache = new ConcurrentHashMap<>();

@@ -35,7 +35,7 @@ public class
DefaultMethodInvokingInterceptor implements
MethodInterceptor {

    InvocationTargetException targetProvider =
(InvocationTargetException) invocation;

    -
        return methodHandleCache.computeIfAbsent(method,
DefaultMethodInvokingInterceptor::lookupMethodHandle)
    +
        return methodHandleCache.computeIfAbsent(method,
DefaultMethodInvoker::lookupMethodHandle)

.bindTo(targetProvider.getInvocationTarget()).invokeWithArgu
ments(invocation getArguments());
}

```

Your task is to identify which refactoring type(s) have been applied in transforming the original program into the new version. Use only the following list of predefined refactorings:

- (Add Method Parameter) – Introduces a new parameter to an existing method.
- (Remove Method Parameter) – Eliminates an existing parameter from a method signature.
- (Rename Method) – Changes the name of a method while preserving its behavior.
- (Rename Class) – Changes the name of a class without altering its structure.
- (Rename Package) – Changes the name of a package declaration.

(Rename Field) – Changes the name of a class or instance variable.

(Extract Class) – Moves a group of related fields and methods from an existing class into a newly created class.

(Extract Superclass) – Creates a new superclass to encapsulate shared attributes and behavior from two or more existing classes.

(Inline Method) – Replaces a method call with the method's body.

(Pull Up Method) – Pull a method from a child class (subclass) in the original code to its parent class (superclass) in the transformed version.

(Push Down Method) – Push a method from a parent class (superclass) in the original code to one or more of its child classes (subclasses) in the transformed version.

(Pull Up Field) – Pull a field from a child class (subclass) in the original code to its parent class (superclass) in the transformed version.

(Push Down Field) – Push a field from a parent class (superclass) in the original code to one or more of its child classes (subclasses) in the transformed version.

(Inline Class) – Merges a class into another when it is too small or redundant.

(Extract Interface) – Creates a new interface from existing method(s) in a class.

(Move Method) – Relocates a method to a more appropriate class.

(Move Field) – Relocates a field to a more appropriate class.

(Replace Magic Number with Constant) – Replaces a literal number with a named constant.

(Encapsulate Field) – Change a field access from not private to private and create get and set methods.

**\*\*Instructions:\*\***

1. Begin your response with a bullet-point list of the refactoring type(s) applied.

2. Then, briefly justify each identified refactoring with reference to the specific code changes.
3. Only include refactorings from the list above.
4. Be concise but precise in your explanations.

Do not generate explanations unrelated to the given transformation.



⋮

- Rename Class

#### Justifications:

- Rename Class:

- The class `DefaultMethodInvokingInterceptor` was renamed to `DefaultMethodInvoker`. This is evident from the file rename operation (`rename from src/main/java/io/lettuce/core/dynamic/intercept/DefaultMethodInvokingInterceptor.java rename to src/main/java/io/lettuce/core/dynamic/intercept/DefaultMethodInvoker.java`) and the change in the class declaration from `public class DefaultMethodInvokingInterceptor` to `public class DefaultMethodInvoker`.
- Consequently, import statements and instantiations of this class were updated in `src/main/java/io/lettuce/core/dynamic/RedisCommandFactory.java` (from `import io.lettuce.core.dynamic.intercept.DefaultMethodInvokingInterceptor;` to `import io.lettuce.core.dynamic.intercept.DefaultMethodInvoker;` and from `new DefaultMethodInvokingInterceptor()` to `new DefaultMethodInvoker()`).
- Additionally, a method reference within the renamed class itself was updated from `DefaultMethodInvokingInterceptor::lookupMethodHandle` to `DefaultMethodInvoker::lookupMethodHandle`.

You are an expert coding assistant specialized in software refactoring, with many years of experience analyzing code

transformations.

You will be given the diffs of a commit:

```
- **Diffs:**  
diff --git  
a/src/main/java/io/lettuce/core/dynamic/DefaultCommandMeth  
odVerifier.java  
b/src/main/java/io/lettuce/core/dynamic/DefaultCommandMet  
hodVerifier.java  
index 1c50fe7ed..0d2f2761d 100644  
----  
a/src/main/java/io/lettuce/core/dynamic/DefaultCommandMeth  
odVerifier.java  
+++  
b/src/main/java/io/lettuce/core/dynamic/DefaultCommandMet  
hodVerifier.java  
@@ -115,15 +115,15 @@ class DefaultCommandMethodVerifier  
implements CommandMethodVerifier {  
    continue;  
}  
  
-      if (bindableParameter.isAssignableTo(KeyValue.class)  
|| bindableParameter.isAssignableTo(ScoredValue.class)) {  
+      if  
(bindableParameter.discoverer.isAssignableTo(KeyValue.class,  
bindableParameter) ||  
bindableParameter.discoverer.isAssignableTo(ScoredValue.cl  
ss, bindableParameter)) {  
    count++;  
}  
  
-      if  
(bindableParameter.isAssignableTo(GeoCoordinates.class) ||  
bindableParameter.isAssignableTo(Range.class)) {
```

```
+      if  
(bindableParameter.discoverer.isAssignableTo(GeoCoordinate  
s.class, bindableParameter) ||  
bindableParameter.discoverer.isAssignableTo(Range.class,  
bindableParameter)) {  
  
    count++;  
}  
  
-      if (bindableParameter.isAssignableTo(Limit.class)) {  
+      if  
(bindableParameter.discoverer.isAssignableTo(Limit.class,  
bindableParameter)) {  
  
    count += 2;  
}  
  
diff --git  
a/src/main/java/io/lettuce/core/dynamic/DefaultMethodParame  
tersAccessor.java  
b/src/main/java/io/lettuce/core/dynamic/DefaultMethodParame  
tersAccessor.java  
index 75ce89334..3e2920712 100644  
---  
a/src/main/java/io/lettuce/core/dynamic/DefaultMethodParame  
tersAccessor.java  
+++  
b/src/main/java/io/lettuce/core/dynamic/DefaultMethodParame  
tersAccessor.java  
@@ -81,9 +81,9 @@ class DefaultMethodParametersAccessor  
implements MethodParametersAccessor {  
  
    Parameter bindableParameter =  
parameters.getBindableParameter(index);  
  
-      if (bindableParameter.isAssignableTo(Limit.class) ||  
bindableParameter.isAssignableTo(io.lettuce.core.Value.class)
```

```
-      ||
bindableParameter.isAssignableTo(KeyValue.class) ||
bindableParameter.isAssignableTo(ScoredVal
ue.class)
-
      ||
bindableParameter.isAssignableTo(GeoCoordinates.class) ||
bindableParameter.isAssignableTo(Range.class)) {
+    if
(bindableParameter.discoverer.isAssignableTo(Limit.class,
bindableParameter) ||
bindableParameter.discoverer.isAssignableTo(io.lettuce.core.V
alue.class, bindableParameter)
+
      ||
bindableParameter.discoverer.isAssignableTo(KeyValue.class,
bindableParameter) ||
bindableParameter.discoverer.isAssignableTo(ScoredValue.cl
ass, bindableParameter)
+
      ||
bindableParameter.discoverer.isAssignableTo(GeoCoordinates
.class, bindableParameter) || bindab
leParameter.discoverer.isAssignableTo(Range.class,
bindableParameter)) {
    return false;
}

diff --git
a/src/main/java/io/lettuce/core/dynamic/parameter/ExecutionS
pecificParameters.java
b/src/main/java/io/lettuce/core/dynamic/parameter/ExecutionS
pecificParameters.java
index 33d18d8f9..b5de752f2 100644
---
a/src/main/java/io/lettuce/core/dynamic/parameter/ExecutionS
pecificParameters.java
+++
b/src/main/java/io/lettuce/core/dynamic/parameter/ExecutionS
```

```
specificParameters.java
@@ -43,11 +43,11 @@ public class
ExecutionSpecificParameters extends
Parameters<ExecutionSpecificPar
    Parameter methodParameter = parameters.get(i);

        if (methodParameter.isSpecialParameter()) {
-            if (methodParameter.isAssignableTo(Timeout.class))
{
+                if
(methodParameter.discoverer.isAssignableTo(Timeout.class,
methodParameter)) {
                    timeoutIndex = i;
}

-                if
(methodParameter.isAssignableTo(CommandBatching.class))
{
+                    if
(methodParameter.discoverer.isAssignableTo(CommandBatch
ing.class, methodParameter)) {
                        commandBatchingIndex = i;
}

diff --git
a/src/main/java/io/lettuce/core/dynamic/parameter/Parameter.j
ava
b/src/main/java/io/lettuce/core/dynamic/parameter/Parameter.j
ava
index 9c0f90968..ac989c6bc 100644
---
a/src/main/java/io/lettuce/core/dynamic/parameter/Parameter.j
ava
+++  
b/src/main/java/io/lettuce/core/dynamic/parameter/Parameter.j
ava
```

```
@@ -6,8 +6,6 @@ import java.util.*;  
import java.util.concurrent.ConcurrentHashMap;  
  
import io.lettuce.core.dynamic.support.*;  
-import io.lettuce.core.internal.LettuceAssert;  
-import io.lettuce.core.internal.LettuceClassUtils;  
  
/**  
 * Abstracts a method parameter and exposes access to type  
 and parameter information.  
@@ -17,7 +15,7 @@ import  
io.lettuce.core.internal.LettuceClassUtils;  
*/  
public class Parameter {  
  
- private final ParameterNameDiscoverer discoverer = new  
CompositeParameterNameDiscoverer()  
+ public final ParameterNameDiscoverer discoverer = new  
CompositeParameterNameDiscoverer(  
    new StandardReflectionParameterNameDiscoverer(),  
new AnnotationParameterNameDiscoverer());  
  
private final Method method;  
@@ -111,19 +109,6 @@ public class Parameter {  
    return typeInformation;  
}  
  
- /**  
- * Check whether the parameter is assignable to {@code target}.  
- *  
- * @param target must not be {@code null}.  
- * @return  
- */  
- public boolean assignableTo(Class<?> target) {  
-  
-
```

```
-    LettuceAssert.notNull(target, "Target type must not be
null");

-
-
-    return LettuceClassUtils.isAssignable(target,
getParameterType());
-
}

-
-
/***
*
* @return {@code true} if the parameter is a special
parameter.

diff --git
a/src/main/java/io/lettuce/core/dynamic/support/ParameterNa
meDiscoverer.java
b/src/main/java/io/lettuce/core/dynamic/support/ParameterNa
meDiscoverer.java
index 609f0caba..aba15b58e 100644
---
a/src/main/java/io/lettuce/core/dynamic/support/ParameterNa
meDiscoverer.java
++++
b/src/main/java/io/lettuce/core/dynamic/support/ParameterNa
meDiscoverer.java
@@ -1,5 +1,9 @@
package io.lettuce.core.dynamic.support;

+import io.lettuce.core.dynamic.parameter.Parameter;
+import io.lettuce.core.internal.LettuceAssert;
+import io.lettuce.core.internal.LettuceClassUtils;
+
import java.lang.reflect.Constructor;
import java.lang.reflect.Method;

@@ -28,4 +32,17 @@ public interface
ParameterNameDiscoverer {

*/
```

```
String[] getParameterNames(Constructor<?> ctor);
```

```
+  /**
+   * Check whether the parameter is assignable to {@code target}.
+   *
+   * @param target must not be {@code null}.
+   * @param parameter
+   * @return
+   */
+ default boolean assignableTo(Class<?> target,
Parameter parameter) {
+
+     LettuceAssert.notNull(target, "Target type must not be
null");
+
+     return LettuceClassUtils.isAssignable(target,
parameter.getParameterType());
+
}
```

Your task is to identify which refactoring type(s) have been applied in transforming the original program into the new version. Use only the following list of predefined refactorings:

(Add Method Parameter) – Introduces a new parameter to an existing method.

(Remove Method Parameter) – Eliminates an existing parameter from a method signature.

(Rename Method) – Changes the name of a method while preserving its behavior.

(Rename Class) – Changes the name of a class without altering its structure.

(Rename Package) – Changes the name of a package declaration.

(Rename Field) – Changes the name of a class or instance variable.

(Extract Class) – Moves a group of related fields and methods from an existing class into a newly created class.

(Extract Superclass) – Creates a new superclass to encapsulate shared attributes and behavior from two or more existing classes.

(Inline Method) – Replaces a method call with the method's body.

(Pull Up Method) – Pull a method from a child class (subclass) in the original code to its parent class (superclass) in the transformed version.

(Push Down Method) – Push a method from a parent class (superclass) in the original code to one or more of its child classes (subclasses) in the transformed version.

(Pull Up Field) – Pull a field from a child class (subclass) in the original code to its parent class (superclass) in the transformed version.

(Push Down Field) – Push a field from a parent class (superclass) in the original code to one or more of its child classes (subclasses) in the transformed version.

(Inline Class) – Merges a class into another when it is too small or redundant.

(Extract Interface) – Creates a new interface from existing method(s) in a class.

(Move Method) – Relocates a method to a more appropriate class.

(Move Field) – Relocates a field to a more appropriate class.

(Replace Magic Number with Constant) – Replaces a literal number with a named constant.

(Encapsulate Field) – Change a field access from not private to private and create get and set methods.

**\*\*Instructions:\*\***

1. Begin your response with a bullet-point list of the refactoring type(s) applied.

2. Then, briefly justify each identified refactoring with reference to the specific code changes.
3. Only include refactorings from the list above.
4. Be concise but precise in your explanations.

Do not generate explanations unrelated to the given transformation.



⋮

- Move Method
- Add Method Parameter

#### Justifications:

- **Move Method:**
  - The method `isAssignableTo(Class<?> target)` was moved from the `Parameter` class to the `ParameterNameDiscoverer` interface (as a new default method).
  - Specifically, the method `public boolean isAssignableTo(Class<?> target)` was removed from  
`src/main/java/io/lettuce/core/dynamic/parameter/Parameter.java`.
  - A new default method `default boolean isAssignableTo(Class<?> target, Parameter parameter)` with the same core logic (checking type assignability using `LettuceClassUtils.isAssignable`) was added to the interface  
`src/main/java/io/lettuce/core/dynamic/support/ParameterNameDiscoverer.java`.
  - Call sites were updated to use the new method through the `discoverer` field of a `Parameter` instance. For example,  
`bindableParameter.isAssignableTo(KeyValue.class)` in `DefaultCommandMethodVerifier.java` became  
`bindableParameter.discoverer.isAssignableTo(KeyValue.class, bindableParameter)`. The field `discoverer` in `Parameter.java` was made public to allow this.
- **Add Method Parameter:**
  - The newly added method `isAssignableTo` in the `ParameterNameDiscoverer` interface has an additional parameter, `Parameter parameter`, compared to the

original method's signature in the `Parameter` class.

- The original method `isAssignableTo(Class<?> target)` in the `Parameter` class implicitly operated on the instance of `Parameter` (i.e., `this`).
- After being moved to `ParameterNameDiscoverer`, the method needed the `Parameter` instance to be passed explicitly to access its `getParameterType()` method (i.e., `parameter.getParameterType()`). Thus, the signature changed from `isAssignableTo(Class<?> target)` to `isAssignableTo(Class<?> target, Parameter parameter)`.

You are an expert coding assistant specialized in software refactoring, with many years of experience analyzing code transformations.

You will be given the diffs of a commit:

```
- **Diffs:**  
diff --git  
a/src/main/java/io/lettuce/core/dynamic/DefaultCommandMethodVerifier.java  
b/src/main/java/io/lettuce/core/dynamic/DefaultCommandMethodVerifier.java  
index dc44640ac..1c50fe7ed 100644  
----  
a/src/main/java/io/lettuce/core/dynamic/DefaultCommandMethodVerifier.java  
+++  
b/src/main/java/io/lettuce/core/dynamic/DefaultCommandMethodVerifier.java  
@@ -11,7 +11,6 @@ import io.lettuce.core.Limit;  
import io.lettuce.core.Range;  
import io.lettuce.core.ScoredValue;  
import io.lettuce.core.dynamic.parameter.Parameter;  
-import io.lettuce.core.dynamic.segment.CommandSegment;  
import io.lettuce.core.dynamic.segment.CommandSegments;
```

```
import io.lettuce.core.internal.LettuceAssert;
import io.lettuce.core.internal.LettuceLists;
@@ -110,7 +109,7 @@ class DefaultCommandMethodVerifier
implements CommandMethodVerifier {

    Parameter bindableParameter =
bindableParameters.get(i);

    -
        boolean consumed =
isConsumed(commandSegments, bindableParameter);
    +
        boolean consumed =
commandSegments.isConsumed(bindableParameter);

    if (consumed) {
        continue;
@@ -134,17 +133,6 @@ class DefaultCommandMethodVerifier
implements CommandMethodVerifier {
        return count;
    }

    -
    -
        private boolean isConsumed(CommandSegments
commandSegments, Parameter bindableParameter) {
    -
    -
        for (CommandSegment commandSegment :
commandSegments) {
    -
        if
            (commandSegment.canConsume(bindableParameter)) {
    -
            return true;
    -
        }
    -
        }
    -
    -
        return false;
    -
}
-
-
private CommandMethodSyntaxException
syntaxException(String commandName, CommandMethod
```

```
commandMethod) {  
  
    CommandMatches commandMatches =  
    CommandMatches.forCommand(commandName,  
    commandDetails);  
    diff --git  
    a/src/main/java/io/lettuce/core/dynamic/segment/CommandSe  
    gments.java  
    b/src/main/java/io/lettuce/core/dynamic/segment/CommandSe  
    gments.java  
    index 5cade513d..6db8b5f79 100644  
    ---  
    a/src/main/java/io/lettuce/core/dynamic/segment/CommandSe  
    gments.java  
    +++  
    b/src/main/java/io/lettuce/core/dynamic/segment/CommandSe  
    gments.java  
    @@ -4,6 +4,7 @@ import java.util.Collections;  
    import java.util.Iterator;  
    import java.util.List;  
  
    +import io.lettuce.core.dynamic.parameter.Parameter;  
    import io.lettuce.core.internal.LettuceAssert;  
    import io.lettuce.core.protocol.CommandType;  
    import io.lettuce.core.protocol.ProtocolKeyword;  
    @@ -64,6 +65,17 @@ public class CommandSegments  
    implements Iterable<CommandSegment> {  
        return segments.size();  
    }  
  
    +    public boolean isConsumed(Parameter  
    bindableParameter) {  
        +  
        +        for (CommandSegment commandSegment : this) {  
        +            if  
        (commandSegment.canConsume(bindableParameter)) {
```

```
+           return true;
+
+       }
+
+   }
+
+       return false;
+   }
+
+
static class StringCommandType implements
ProtocolKeyword {

    private final byte[] commandTypeBytes;
```

Your task is to identify which refactoring type(s) have been applied in transforming the original program into the new version. Use only the following list of predefined refactorings:

(Add Method Parameter) – Introduces a new parameter to an existing method.

(Remove Method Parameter) – Eliminates an existing parameter from a method signature.

(Rename Method) – Changes the name of a method while preserving its behavior.

(Rename Class) – Changes the name of a class without altering its structure.

(Rename Package) – Changes the name of a package declaration.

(Rename Field) – Changes the name of a class or instance variable.

(Extract Class) – Moves a group of related fields and methods from an existing class into a newly created class.

(Extract Superclass) – Creates a new superclass to encapsulate shared attributes and behavior from two or more existing classes.

(Inline Method) – Replaces a method call with the method's body.

(Pull Up Method) – Pull a method from a child class (subclass) in the original code to its parent class (superclass) in the transformed version.

(Push Down Method) – Push a method from a parent class (superclass) in the original code to one or more of its child classes (subclasses) in the transformed version.

(Pull Up Field) – Pull a field from a child class (subclass) in the original code to its parent class (superclass) in the transformed version.

(Push Down Field) – Push a field from a parent class (superclass) in the original code to one or more of its child classes (subclasses) in the transformed version.

(Inline Class) – Merges a class into another when it is too small or redundant.

(Extract Interface) – Creates a new interface from existing method(s) in a class.

(Move Method) – Relocates a method to a more appropriate class.

(Move Field) – Relocates a field to a more appropriate class.

(Replace Magic Number with Constant) – Replaces a literal number with a named constant.

(Encapsulate Field) – Change a field access from not private to private and create get and set methods.

**\*\*Instructions:\*\***

1. Begin your response with a bullet-point list of the refactoring type(s) applied.
2. Then, briefly justify each identified refactoring with reference to the specific code changes.
3. Only include refactorings from the list above.
4. Be concise but precise in your explanations.

Do not generate explanations unrelated to the given transformation.



- Move Method
- Remove Method Parameter

#### Justifications:

- **Move Method:**
  - The method `isConsumed` was moved from the `DefaultCommandMethodVerifier` class to the `CommandSegments` class.
  - In `DefaultCommandMethodVerifier.java`, the private method `isConsumed(CommandSegments commandSegments, Parameter bindableParameter)` was removed.
  - A new public method `isConsumed(Parameter bindableParameter)` with identical logic (iterating over its own segments and calling `commandSegment.canConsume()`) was added to `CommandSegments.java`.
  - The call site in `DefaultCommandMethodVerifier.java` was changed from `isConsumed(commandSegments, bindableParameter)` to `commandSegments.isConsumed(bindableParameter)`, reflecting the method's new location. This move is appropriate as the method's logic primarily operates on the data within `CommandSegments`.
- **Remove Method Parameter:**
  - When the `isConsumed` method was moved to the `CommandSegments` class, its parameter `CommandSegments commandSegments` was removed.
  - The original signature in `DefaultCommandMethodVerifier` was `isConsumed(CommandSegments commandSegments, Parameter bindableParameter)`.
  - The new signature in `CommandSegments` is `isConsumed(Parameter bindableParameter)`.
  - The `commandSegments` parameter became redundant because the method, now being part of the `CommandSegments` class, can directly access the command segments of the instance (`this`) on which it is called.

You are an expert coding assistant specialized in software refactoring, with many years of experience analyzing code transformations.

You will be given the diffs of a commit:

```
- **Diffs:**  
diff --git  
a/src/main/java/io/lettuce/core/dynamic/output/CommandOutputResolverSupport.java  
b/src/main/java/io/lettuce/core/dynamic/output/CommandOutputResolverSupport.java  
index 367f1ab79..6f09fecb9 100644  
----  
a/src/main/java/io/lettuce/core/dynamic/output/CommandOutputResolverSupport.java  
+++  
b/src/main/java/io/lettuce/core/dynamic/output/CommandOutputResolverSupport.java  
@@ -1,5 +1,7 @@  
package io.lettuce.core.dynamic.output;  
  
+import io.lettuce.core.dynamic.support.ResolvableType;  
+  
/**  
 * Base class for {@link CommandOutputFactory} resolution  
such as {@link  
OutputRegistryCommandOutputFactoryResolver}.  
* <p>  
@@ -10,4 +12,20 @@ package io.lettuce.core.dynamic.output;  
*/  
public abstract class CommandOutputResolverSupport {  
  
+    /**  
+     * Overridable hook to check whether {@code selector}  
can be assigned from the provider type {@code provider}  
.        .  
+     * <p>
```

```
+ * This method descends the component type hierarchy
+ and considers primitive/wrapper type conversion.
+
+ *
+ * @param selector must not be {@code null}.
+ * @param provider must not be {@code null}.
+ * @return {@code true} if selector can be assigned from
+ its provider type.
+ */
+ protected boolean assignableFrom(OutputSelector
+ selector, OutputType provider) {
+
+ ResolvableType selectorType =
+ selector.getOutputType();
+ ResolvableType resolvableType =
+ provider.withCodec(selector.getRedisCodec());
+
+ return selectorType.isAssignableFrom(resolvableType);
+ }
}
diff --git
a/src/main/java/io/lettuce/core/dynamic/output/OutputRegistry
CommandOutputFactoryResolver.java b/src/ma
in/java/io/lettuce/core/dynamic/output/OutputRegistryComma
ndOutputFactoryResolver.java
index 6019929a3..8ba3c724a 100644
---
a/src/main/java/io/lettuce/core/dynamic/output/OutputRegistry
CommandOutputFactoryResolver.java
+++
b/src/main/java/io/lettuce/core/dynamic/output/OutputRegistry
CommandOutputFactoryResolver.java
@@ -6,7 +6,6 @@ import java.util.Map;
import java.util.stream.Collectors;

import io.lettuce.core.dynamic.support.ClassTypeInformation;
-import io.lettuce.core.dynamic.support.ResolvableType;
```

```

import io.lettuce.core.internal.LettuceAssert;
import io.lettuce.core.output.CommandOutput;

@@ -88,20 +87,4 @@ public class
OutputRegistryCommandOutputFactoryResolver extends
CommandOutputRes
}).collect(Collectors.toList());
}

-
- /**
- * Overridable hook to check whether {@code selector}
can be assigned from the provider type {@code provider}.
- * <p>
- * This method descends the component type hierarchy
and considers primitive/wrapper type conversion.
- *
- * @param selector must not be {@code null}.
- * @param provider must not be {@code null}.
- * @return {@code true} if selector can be assigned from
its provider type.
- */
- protected boolean assignableFrom(OutputSelector
selector, OutputType provider) {
-
- ResolvableType selectorType =
selector.getOutputType();
- ResolvableType resolvableType =
provider.withCodec(selector.getRedisCodec());
-
- return selectorType.isAssignableFrom(resolvableType);
- }
}

```

Your task is to identify which refactoring type(s) have been applied in transforming the original program into the new version. Use only the following list of predefined refactorings:

- (Add Method Parameter) – Introduces a new parameter to an existing method.
- (Remove Method Parameter) – Eliminates an existing parameter from a method signature.
- (Rename Method) – Changes the name of a method while preserving its behavior.
- (Rename Class) – Changes the name of a class without altering its structure.
- (Rename Package) – Changes the name of a package declaration.
- (Rename Field) – Changes the name of a class or instance variable.
- (Extract Class) – Moves a group of related fields and methods from an existing class into a newly created class.
- (Extract Superclass) – Creates a new superclass to encapsulate shared attributes and behavior from two or more existing classes.
- (Inline Method) – Replaces a method call with the method's body.
- (Pull Up Method) – Pull a method from a child class (subclass) in the original code to its parent class (superclass) in the transformed version.
- (Push Down Method) – Push a method from a parent class (superclass) in the original code to one or more of its child classes (subclasses) in the transformed version.
- (Pull Up Field) – Pull a field from a child class (subclass) in the original code to its parent class (superclass) in the transformed version.
- (Push Down Field) – Push a field from a parent class (superclass) in the original code to one or more of its child classes (subclasses) in the transformed version.
- (Inline Class) – Merges a class into another when it is too small or redundant.
- (Extract Interface) – Creates a new interface from existing method(s) in a class.

(Move Method) – Relocates a method to a more appropriate class.

(Move Field) – Relocates a field to a more appropriate class.

(Replace Magic Number with Constant) – Replaces a literal number with a named constant.

(Encapsulate Field) – Change a field access from not private to private and create get and set methods.

**\*\*Instructions:\*\***

1. Begin your response with a bullet-point list of the refactoring type(s) applied.
2. Then, briefly justify each identified refactoring with reference to the specific code changes.
3. Only include refactorings from the list above.
4. Be concise but precise in your explanations.

Do not generate explanations unrelated to the given transformation.



- Pull Up Method

#### Justifications:

- **Pull Up Method:**

- The method `protected boolean assignableFrom(OutputSelector selector, OutputType provider)` was moved from the child class `OutputRegistryCommandOutputFactoryResolver` to its parent class `CommandOutputResolverSupport`.
- In `OutputRegistryCommandOutputFactoryResolver.java` (the subclass), this method was removed.
- An identical method (same signature, same body, and same visibility) was added to `CommandOutputResolverSupport.java` (the superclass).
- This refactoring is used to move common behavior from subclasses to a shared superclass.

You are an expert coding assistant specialized in software refactoring, with many years of experience analyzing code transformations.

You will be given the diffs of a commit:

```
- **Diffs:**  
diff --git  
a/src/main/java/io/lettuce/core/AbstractRedisClient.java  
b/src/main/java/io/lettuce/core/AbstractRedisClient.java  
index 5dc349fd4..96f8fcd88 100644  
--- a/src/main/java/io/lettuce/core/AbstractRedisClient.java  
+++ b/src/main/java/io/lettuce/core/AbstractRedisClient.java  
@@ -621,4 +621,17 @@ public abstract class  
AbstractRedisClient implements AutoCloseable {  
    state);  
}  
  
+ /**  
+ * Add a listener for the RedisConnectionState. The listener  
is notified every time a connect/disconnect/IO exception  
+ * happens. The listeners are not bound to a specific  
connection, so every time a connection event happens on any  
+ * connection, the listener will be notified. The  
corresponding netty channel handler (async connection) is p  
assed on the  
+ * event.  
+ *  
+ * @param listener must not be {@code null}.  
+ */  
+ public void addListener(RedisConnectionStateListener  
listener) {  
+
```

```

+     LettuceAssert.notNull(listener,
"RedisConnectionStateListener must not be null");
+     connectionEvents.addListener(listener);
+   }
}

diff --git a/src/main/java/io/lettuce/core/RedisClient.java
b/src/main/java/io/lettuce/core/RedisClient.java
index 371a90242..78b28d3a1 100644
--- a/src/main/java/io/lettuce/core/RedisClient.java
+++ b/src/main/java/io/lettuce/core/RedisClient.java
@@ -841,17 +841,4 @@ public class RedisClient extends
AbstractRedisClient {
    checkValidRedisURI(this.redisURI);
}

-
- /**
- * Add a listener for the RedisConnectionState. The listener
is notified every time a connect/disconnect/IO exception
- * happens. The listeners are not bound to a specific
connection, so every time a connection event happens on any
- * connection, the listener will be notified. The
corresponding netty channel handler (async connection) is
passed on the
- * event.
- *
- * @param listener must not be {@code null}.
- */
-
- public void addListener(RedisConnectionStateListener
listener) {
-
-
-     LettuceAssert.notNull(listener,
"RedisConnectionStateListener must not be null");
-     connectionEvents.addListener(listener);
-   }
}

```

```

diff --git
a/src/main/java/io/lettuce/core/cluster/RedisClusterClient.java
b/src/main/java/io/lettuce/core/cluster/RedisClusterClient.java
index c37bbca54..f48412c23 100644
---
a/src/main/java/io/lettuce/core/cluster/RedisClusterClient.java
+++  

b/src/main/java/io/lettuce/core/cluster/RedisClusterClient.java
@@ -1341,8 +1341,7 @@ public class RedisClusterClient
extends AbstractRedisClient {
    */
    public void addListener(RedisConnectionStateListener
listener) {
  

-        LettuceAssert.notNull(listener,
"RedisConnectionStateListener must not be null");
-        connectionEvents.addListener(listener);
+        super.addListener(listener);
    }
  

private class NodeConnectionFactoryImpl implements
NodeConnectionFactory {

```

Your task is to identify which refactoring type(s) have been applied in transforming the original program into the new version. Use only the following list of predefined refactorings:

(Add Method Parameter) – Introduces a new parameter to an existing method.

(Remove Method Parameter) – Eliminates an existing parameter from a method signature.

(Rename Method) – Changes the name of a method while preserving its behavior.

(Rename Class) – Changes the name of a class without altering its structure.

(Rename Package) – Changes the name of a package declaration.

(Rename Field) – Changes the name of a class or instance variable.

(Extract Class) – Moves a group of related fields and methods from an existing class into a newly created class.

(Extract Superclass) – Creates a new superclass to encapsulate shared attributes and behavior from two or more existing classes.

(Inline Method) – Replaces a method call with the method's body.

(Pull Up Method) – Pull a method from a child class (subclass) in the original code to its parent class (superclass) in the transformed version.

(Push Down Method) – Push a method from a parent class (superclass) in the original code to one or more of its child classes (subclasses) in the transformed version.

(Pull Up Field) – Pull a field from a child class (subclass) in the original code to its parent class (superclass) in the transformed version.

(Push Down Field) – Push a field from a parent class (superclass) in the original code to one or more of its child classes (subclasses) in the transformed version.

(Inline Class) – Merges a class into another when it is too small or redundant.

(Extract Interface) – Creates a new interface from existing method(s) in a class.

(Move Method) – Relocates a method to a more appropriate class.

(Move Field) – Relocates a field to a more appropriate class.

(Replace Magic Number with Constant) – Replaces a literal number with a named constant.

(Encapsulate Field) – Change a field access from not private to private and create get and set methods.

**\*\*Instructions:\*\***

1. Begin your response with a bullet-point list of the refactoring type(s) applied.
2. Then, briefly justify each identified refactoring with reference to the specific code changes.
3. Only include refactorings from the list above.
4. Be concise but precise in your explanations.

Do not generate explanations unrelated to the given transformation.



⋮

- Pull Up Method

#### Justifications:

- **Pull Up Method:**

- The common implementation of the method `public void addListener(RedisConnectionStateListener listener)` was moved from the subclasses (`RedisClient` and effectively `RedisClusterClient`) to their shared superclass `AbstractRedisClient`.
- In `RedisClient.java`, the `addListener` method, which had the implementation `LettuceAssert.notNull(listener, "RedisConnectionStateListener must not be null"); connectionEvents.addListener(listener);`, was removed entirely, indicating it now inherits this behavior from `AbstractRedisClient`.
- In `AbstractRedisClient.java`, this exact method (same signature, body, and visibility) was added.
- In `RedisClusterClient.java`, the `addListener` method, which previously had the same implementation, was modified to delegate to the superclass's new method by changing its body to `super.addListener(listener);`. This shows that the common logic was centralized in the superclass.
- This refactoring consolidates duplicate code from subclasses into a single place in the superclass.

You are an expert coding assistant specialized in software refactoring, with many years of experience analyzing code transformations.

You will be given the diffs of a commit:

```
- **Diffs:**  
diff --git  
a/src/main/java/io/lettuce/core/dynamic/output/CommandOutputResolverSupport.java  
b/src/main/java/io/lettuce/core/dynamic/output/CommandOutputResolverSupport.java  
index 448d987d0..367f1ab79 100644  
---  
a/src/main/java/io/lettuce/core/dynamic/output/CommandOutputResolverSupport.java  
+++  
b/src/main/java/io/lettuce/core/dynamic/output/CommandOutputResolverSupport.java  
@@ -1,7 +1,5 @@  
package io.lettuce.core.dynamic.output;  
  
-import io.lettuce.core.dynamic.support.ResolvableType;  
-  
/**  
 * Base class for {@link CommandOutputFactory} resolution  
such as {@link  
OutputRegistryCommandOutputFactoryResolver}.  
* <p>  
@@ -12,21 +10,4 @@ import  
io.lettuce.core.dynamic.support.ResolvableType;  
*/  
public abstract class CommandOutputResolverSupport {  
  
- /**
```

```
- * Overridable hook to check whether {@code selector}
can be assigned from the provider type {@code provider}
-
- * <p>
- * This method descends the component type hierarchy
and considers primitive/wrapper type conversion.
- *
- * @param selector must not be {@code null}.
- * @param provider must not be {@code null}.
- * @return {@code true} if selector can be assigned from
its provider type.
- */
- protected boolean assignableFrom(OutputSelector
selector, OutputType provider) {
-
- ResolvableType selectorType =
selector.getOutputType();
- ResolvableType resolvableType =
provider.withCodec(selector.getRedisCodec());
-
- return selectorType.isAssignableFrom(resolvableType);
- }
-
}
diff --git
a/src/main/java/io/lettuce/core/dynamic/output/OutputRegistry
CommandOutputFactoryResolver.java
b/src/main/java/io/lettuce/core/dynamic/output/OutputRegistry
CommandOutputFactoryResolver.java
index 8ba3c724a..6019929a3 100644
---
a/src/main/java/io/lettuce/core/dynamic/output/OutputRegistry
CommandOutputFactoryResolver.java
+++
b/src/main/java/io/lettuce/core/dynamic/output/OutputRegistry
CommandOutputFactoryResolver.java
```

```
@@ -6,6 +6,7 @@ import java.util.Map;
import java.util.stream.Collectors;

import io.lettuce.core.dynamic.support.ClassTypeInformation;
+import io.lettuce.core.dynamic.support.ResolvableType;
import io.lettuce.core.internal.LettuceAssert;
import io.lettuce.core.output.CommandOutput;

@@ -87,4 +88,20 @@ public class
OutputRegistryCommandOutputFactoryResolver extends
CommandOutputRes
}).collect(Collectors.toList());
}

+
+ /**
+ * Overridable hook to check whether {@code selector}
can be assigned from the provider type {@code provider}.
+ * <p>
+ * This method descends the component type hierarchy
and considers primitive/wrapper type conversion.
+ *
+ * @param selector must not be {@code null}.
+ * @param provider must not be {@code null}.
+ * @return {@code true} if selector can be assigned from
its provider type.
+ */
+ protected boolean assignableFrom(OutputSelector
selector, OutputType provider) {
+
+     ResolvableType selectorType =
selector.getOutputType();
+     ResolvableType resolvableType =
provider.withCodec(selector.getRedisCodec());
+
+     return selectorType.isAssignableFrom(resolvableType);
+ }
```

}

Your task is to identify which refactoring type(s) have been applied in transforming the original program into the new version. Use only the following list of predefined refactorings:

- (Add Method Parameter) – Introduces a new parameter to an existing method.
- (Remove Method Parameter) – Eliminates an existing parameter from a method signature.
- (Rename Method) – Changes the name of a method while preserving its behavior.
- (Rename Class) – Changes the name of a class without altering its structure.
- (Rename Package) – Changes the name of a package declaration.
- (Rename Field) – Changes the name of a class or instance variable.
- (Extract Class) – Moves a group of related fields and methods from an existing class into a newly created class.
- (Extract Superclass) – Creates a new superclass to encapsulate shared attributes and behavior from two or more existing classes.
- (Inline Method) – Replaces a method call with the method's body.
- (Pull Up Method) – Pull a method from a child class (subclass) in the original code to its parent class (superclass) in the transformed version.
- (Push Down Method) – Push a method from a parent class (superclass) in the original code to one or more of its child classes (subclasses) in the transformed version.
- (Pull Up Field) – Pull a field from a child class (subclass) in the original code to its parent class (superclass) in the transformed version.
- (Push Down Field) – Push a field from a parent class (superclass) in the original code to one or more of its child

classes (subclasses) in the transformed version.

(Inline Class) – Merges a class into another when it is too small or redundant.

(Extract Interface) – Creates a new interface from existing method(s) in a class.

(Move Method) – Relocates a method to a more appropriate class.

(Move Field) – Relocates a field to a more appropriate class.

(Replace Magic Number with Constant) – Replaces a literal number with a named constant.

(Encapsulate Field) – Change a field access from not private to private and create get and set methods.

**\*\*Instructions:\*\***

1. Begin your response with a bullet-point list of the refactoring type(s) applied.
2. Then, briefly justify each identified refactoring with reference to the specific code changes.
3. Only include refactorings from the list above.
4. Be concise but precise in your explanations.

Do not generate explanations unrelated to the given transformation.



- Push Down Method

#### Justifications:

- Push Down Method:

- The method `protected boolean assignableFrom(OutputSelector selector, OutputType provider)` was moved from the parent class `CommandOutputResolverSupport` to its child class `OutputRegistryCommandOutputFactoryResolver`.
- In `CommandOutputResolverSupport.java` (the superclass), this method was removed.

- An identical method (same signature, same body, and same visibility) was added to `OutputRegistryCommandOutputFactoryResolver.java` (the subclass).
- This refactoring is used when a method in a superclass is only relevant to one or a few subclasses, moving it closer to where it is actually used.

```
diff --git
a/src/main/java/io/lettuce/core/AbstractRedisClient.java
b/src/main/java/io/lettuce/core/AbstractRedisClient.java
index 9adff5062..5dc349fd4 100644
--- a/src/main/java/io/lettuce/core/AbstractRedisClient.java
+++ b/src/main/java/io/lettuce/core/AbstractRedisClient.java
@@ -212,20 +212,6 @@ public abstract class
AbstractRedisClient implements AutoCloseable {
    return closeableResources.size();
}

-
- /**
- * Add a listener for the RedisConnectionState. The listener
is notified every time a connect/disconnect/IO exception
- * happens. The listeners are not bound to a specific
connection, so every time a connection event happens on any
- * connection, the listener will be notified. The
corresponding netty channel handler (async connection) is p
assed on the
- * event.
- *
- * @param listener must not be {@code null}.
- */
- public void addListener(RedisConnectionStateListener
listener) {
-
-     LettuceAssert.notNull(listener,
"RedisConnectionStateListener must not be null");
-     connectionEvents.addListener(listener);
- }
```

```
-  
    /**  
     * Removes a listener.  
     *  
diff --git a/src/main/java/io/lettuce/core/RedisClient.java  
b/src/main/java/io/lettuce/core/RedisClient.java  
index 78b28d3a1..371a90242 100644  
--- a/src/main/java/io/lettuce/core/RedisClient.java  
+++ b/src/main/java/io/lettuce/core/RedisClient.java  
@@ -841,4 +841,17 @@ public class RedisClient extends  
AbstractRedisClient {  
    checkValidRedisURI(this.redisURI);  
}  
  
+  /**  
+   * Add a listener for the RedisConnectionState. The listener  
is notified every time a connect/disconnect/IO exception  
+   * happens. The listeners are not bound to a specific  
connection, so every time a connection event happens on any  
+   * connection, the listener will be notified. The  
corresponding netty channel handler (async connection) is p  
assed on the  
+   * event.  
+   *  
+   * @param listener must not be {@code null}.  
+   */  
+  public void addListener(RedisConnectionStateListener  
listener) {  
  
+  
+    LettuceAssert.notNull(listener,  
"RedisConnectionStateListener must not be null");  
+    connectionEvents.addListener(listener);  
+  }  
}  
diff --git  
a/src/main/java/io/lettuce/core/cluster/RedisClusterClient.java
```

```
b/src/main/java/io/lettuce/core/cluster/RedisClusterClient.java
index b5fa5cf19..c37bbca54 100644
---
a/src/main/java/io/lettuce/core/cluster/RedisClusterClient.java
+++  
b/src/main/java/io/lettuce/core/cluster/RedisClusterClient.java
@@ -1331,6 +1331,20 @@ public class RedisClusterClient
extends AbstractRedisClient {
    LettuceAssert.notNull(clientResources, "ClientResources
must not be null");
}

+ /**
+ * Add a listener for the RedisConnectionState. The listener
is notified every time a connect/disconnect/IO exception
+ * happens. The listeners are not bound to a specific
connection, so every time a connection event happens on any
+ * connection, the listener will be notified. The
corresponding netty channel handler (async connection) is
passed on the
+ * event.
+
+ * @param listener must not be {@code null}.
+
+ */
+ public void addListener(RedisConnectionStateListener
listener) {
+
+     LettuceAssert.notNull(listener,
"RedisConnectionStateListener must not be null");
+     connectionEvents.addListener(listener);
+
}
+
private class NodeConnectionFactoryImpl implements
NodeConnectionFactory {

    @Override
```



- Push Down Method

#### Justifications:

- Push Down Method:

- The method `public void addListener(RedisConnectionStateListener listener)` was moved from the superclass `AbstractRedisClient` to its subclasses `RedisClient` and `RedisClusterClient`.
- In `AbstractRedisClient.java`, the `addListener` method was removed.
- An identical method (same signature, same body):  
`LettuceAssert.notNull(listener, "RedisConnectionStateListener must not be null"); connectionEvents.addListener(listener);`, and same visibility) was added to both `RedisClient.java` and `RedisClusterClient.java`.
- This refactoring is often used when a method in a superclass is not used by all subclasses, or when subclasses need different implementations (though in this specific diff, the implementations pushed down are identical).

```
diff --git a/src/main/java/io/lettuce/core/ILettuceVersion.java b/src/main/java/io/lettuce/core/ILettuceVersion.java new file mode 100644 index 00000000..67f98fb60 --- /dev/null +++ b/src/main/java/io/lettuce/core/ILettuceVersion.java @@ -0,0 +1,20 @@ +package io.lettuce.core; + + +public class ILettuceVersion { +    /** +     * Return the library name. +     */ +    public static String getName() { +        return "Lettuce"; +    } }
```

```

+  }

+
+ /**
+  * Return the full version string of the present Lettuce
+  codebase, or {@code null} if it cannot be determined.
+ *
+  * @see Package#getImplementationVersion()
+ */
+ public static String getVersion() {
+     Package pkg = LettuceVersion.class.getPackage();
+     return (pkg != null ? pkg.getImplementationVersion() :
null);
+ }
+}

diff --git a/src/main/java/io/lettuce/core/LettuceVersion.java
b/src/main/java/io/lettuce/core/LettuceVersion.java index
8b7f9bb78..8edee0b43 100644
--- a/src/main/java/io/lettuce/core/LettuceVersion.java
+++ b/src/main/java/io/lettuce/core/LettuceVersion.java
@@ -10,26 +10,9 @@ package io.lettuce.core;
 * @author Mark Paluch
 * @since 6.3
 */
-public final class LettuceVersion {
+public final class LettuceVersion extends ILettuceVersion {

    private LettuceVersion() {
    }

-
- /**
-  * Return the library name.
- */
- public static String getName() {
-     return "Lettuce";
- }
-
```

```
- /**
- * Return the full version string of the present Lettuce
- * codebase, or {@code null} if it cannot be determined.
- *
- * @see Package#getImplementationVersion()
- */
- public static String getVersion() {
-     Package pkg = LettuceVersion.class.getPackage();
-     return (pkg != null ? pkg.getImplementationVersion() :
-             null);
- }
-
```



⋮

- Extract Superclass

#### Justifications:

- Extract Superclass:
  - A new class `ILettuceVersion` was created (new file `src/main/java/io/lettuce/core/ILettuceVersion.java`).
  - The static methods `getName()` and `getVersion()` were removed from the existing class `LettuceVersion`.
  - These exact methods (same signature, body, and static nature) were added to the new `ILettuceVersion` class.
  - The class `LettuceVersion` was modified to extend the new `ILettuceVersion` class (`public final class LettuceVersion extends ILettuceVersion`).
  - This refactoring pattern involves creating a new superclass and moving common members (fields or methods) from one or more existing classes to this new superclass, then making the original class(es) inherit from it. In this case, functionality was extracted from `LettuceVersion` into `ILettuceVersion`.

You are an expert coding assistant specialized in software refactoring, with many years of experience analyzing code transformations.

You will be given the diffs of a commit:

```
- **Diffs:**  
diff --git  
a/src/main/java/io/lettuce/core/dynamic/ConversionService.jav  
a  
b/src/main/java/io/lettuce/core/dynamic/ConversionService.jav  
a  
index 501bb7a95..e4ebd316e 100644  
---  
a/src/main/java/io/lettuce/core/dynamic/ConversionService.jav  
a  
+++  
b/src/main/java/io/lettuce/core/dynamic/ConversionService.jav  
a  
@@ -13,7 +13,7 @@ import  
io.lettuce.core.internal.LettuceAssert;  
/**  
 * @author Mark Paluch  
 */  
-class ConversionService {  
+class ConversionService implements IConversionService {  
  
    private Map<ConvertiblePair, Function<?, ?>> converterMap  
= new HashMap<>(10);  
  
@@ -23,6 +23,7 @@ class ConversionService {  
    * @param converter the converter.  
    */  
    @SuppressWarnings("rawtypes")  
+    @Override
```

```
public void addConverter(Function<?, ?> converter) {  
  
    LettuceAssert.notNull(converter, "Converter must not be  
    null");  
    @@ -36,6 +37,7 @@ class ConversionService {  
        }  
  
        @SuppressWarnings("unchecked")  
        +  @Override  
        public <S, T> T convert(S source, Class<T> targetType) {  
  
            LettuceAssert.notNull(source, "Source must not be  
            null");  
            @@ -43,6 +45,7 @@ class ConversionService {  
                return (T) getConverter(source.getClass(),  
                targetType).apply(source);  
            }  
  
            +  @Override  
            public <S, T> boolean canConvert(Class<S> sourceType,  
            Class<T> targetType) {  
                return findConverter(sourceType,  
                targetType).isPresent();  
            }  
diff --git  
a/src/main/java/io/lettuce/core/dynamic/IConversionService.jav  
a  
b/src/main/java/io/lettuce/core/dynamic/IConversionService.jav  
a  
new file mode 100644  
index 00000000..b45669cb1  
--- /dev/null  
+++  
b/src/main/java/io/lettuce/core/dynamic/IConversionService.jav  
a  
@@ -0,0 +1,13 @@
```

```
+package io.lettuce.core.dynamic;  
+  
+import java.util.function.Function;  
+  
+public interface IConversionService {  
+    @SuppressWarnings("rawtypes")  
+    void addConverter(Function<?, ?> converter);  
+  
+    @SuppressWarnings("unchecked")  
+    <S, T> T convert(S source, Class<T> targetType);  
+  
+    <S, T> boolean canConvert(Class<S> sourceType, Class<T>  
targetType);  
+}
```

Your task is to identify which refactoring type(s) have been applied in transforming the original program into the new version. Use only the following list of predefined refactorings:

(Add Method Parameter) – Introduces a new parameter to an existing method.

(Remove Method Parameter) – Eliminates an existing parameter from a method signature.

(Rename Method) – Changes the name of a method while preserving its behavior.

(Rename Class) – Changes the name of a class without altering its structure.

(Rename Package) – Changes the name of a package declaration.

(Rename Field) – Changes the name of a class or instance variable.

(Extract Class) – Moves a group of related fields and methods from an existing class into a newly created class.

(Extract Superclass) – Creates a new superclass to encapsulate shared attributes and behavior from two or more existing classes.

(Inline Method) – Replaces a method call with the method's body.

(Pull Up Method) – Pull a method from a child class (subclass) in the original code to its parent class (superclass) in the transformed version.

(Push Down Method) – Push a method from a parent class (superclass) in the original code to one or more of its child classes (subclasses) in the transformed version.

(Pull Up Field) – Pull a field from a child class (subclass) in the original code to its parent class (superclass) in the transformed version.

(Push Down Field) – Push a field from a parent class (superclass) in the original code to one or more of its child classes (subclasses) in the transformed version.

(Inline Class) – Merges a class into another when it is too small or redundant.

(Extract Interface) – Creates a new interface from existing method(s) in a class.

(Move Method) – Relocates a method to a more appropriate class.

(Move Field) – Relocates a field to a more appropriate class.

(Replace Magic Number with Constant) – Replaces a literal number with a named constant.

(Encapsulate Field) – Change a field access from not private to private and create get and set methods.

**\*\*Instructions:\*\***

1. Begin your response with a bullet-point list of the refactoring type(s) applied.
2. Then, briefly justify each identified refactoring with reference to the specific code changes.
3. Only include refactorings from the list above.
4. Be concise but precise in your explanations.

Do not generate explanations unrelated to the given transformation.



- Extract Interface

#### Justifications:

- Extract Interface:

- A new interface, `IConversionService`, was created (new file `src/main/java/io/lettuce/core/dynamic/IConversionService.java`).
- This new interface declares methods (`addConverter`, `convert`, `canConvert`) whose signatures match existing public methods in the `ConversionService` class.
- The existing class `ConversionService` was modified to implement this new `IConversionService` interface (its declaration changed from `class ConversionService` to `class ConversionService implements IConversionService`).
- The methods in `ConversionService` that correspond to the interface methods are now marked with the `@Override` annotation.
- This refactoring allows for defining a common type for classes that provide conversion services, promoting abstraction and enabling polymorphism.

You are an expert coding assistant specialized in software refactoring, with many years of experience analyzing code transformations.

You will be given the diffs of a commit:

```
- **Diffs:**  
diff --git  
src/main/java/io/lettuce/core/resource/AddressResolverGroup  
Provider.java  
src/main/java/io/lettuce/core/resource/AddressResolverGroup  
Provider.java  
index effa2e50..1b59d949 100644
```

```
---
```

```
src/main/java/io/lettuce/core/resource/AddressResolverGroup
Provider.java
```

```
+++
```

```
src/main/java/io/lettuce/core/resource/AddressResolverGroup
Provider.java
```

```
@@ -20,9 +20,7 @@ import
io.netty.util.internal.logging.InternalLoggerFactory;
```

```
 * @author Mark Paluch
 * @since 6.1
 */
```

```
-class AddressResolverGroupProvider {
```

```
-
```

```
- private static final InternalLogger logger =
InternalLoggerFactory.getInstance(AddressResolverGroupPro
vider.class);
```

```
+class AddressResolverGroupProvider extends
EAddressResolverGroupProvider {
```

```
    private static final AddressResolverGroup<?>
ADDRESS_RESOLVER_GROUP;
```

```
diff --git
src/main/java/io/lettuce/core/resource/EAddressResolverGrou
pProvider.java
src/main/java/io/lettuce/core/resource/EAddressResolverGrou
pProvider.java
new file mode 100644
index 0000000..0bbf5b2f
--- /dev/null
+++
```

```
src/main/java/io/lettuce/core/resource/EAddressResolverGrou
pProvider.java
```

```
@@ -0,0 +1,10 @@
+package io.lettuce.core.resource;
```

```
+
```

```
+import io.netty.util.internal.logging.InternalLogger;
+import io.netty.util.internal.logging.InternalLoggerFactory;
+
+public class EAddressResolverGroupProvider {
+
+    protected static final InternalLogger logger =
InternalLoggerFactory.getInstance(AddressResolverGroupProvider.class);
+
+}
```

Your task is to identify which refactoring type(s) have been applied in transforming the original program into the new version. Use only the following list of predefined refactorings:

(Add Method Parameter) – Introduces a new parameter to an existing method.

(Remove Method Parameter) – Eliminates an existing parameter from a method signature.

(Rename Method) – Changes the name of a method while preserving its behavior.

(Rename Class) – Changes the name of a class without altering its structure.

(Rename Package) – Changes the name of a package declaration.

(Rename Field) – Changes the name of a class or instance variable.

(Extract Class) – Moves a group of related fields and methods from an existing class into a newly created class.

(Extract Superclass) – Creates a new superclass to encapsulate shared attributes and behavior from two or more existing classes.

(Inline Method) – Replaces a method call with the method's body.

(Pull Up Method) – Pull a method from a child class (subclass) in the original code to its parent class (superclass) in the transformed version.

(Push Down Method) – Push a method from a parent class (superclass) in the original code to one or more of its child classes (subclasses) in the transformed version.

(Pull Up Field) – Pull a field from a child class (subclass) in the original code to its parent class (superclass) in the transformed version.

(Push Down Field) – Push a field from a parent class (superclass) in the original code to one or more of its child classes (subclasses) in the transformed version.

(Inline Class) – Merges a class into another when it is too small or redundant.

(Extract Interface) – Creates a new interface from existing method(s) in a class.

(Move Method) – Relocates a method to a more appropriate class.

(Move Field) – Relocates a field to a more appropriate class.

(Replace Magic Number with Constant) – Replaces a literal number with a named constant.

(Encapsulate Field) – Change a field access from not private to private and create get and set methods.

**\*\*Instructions:\*\***

1. Begin your response with a bullet-point list of the refactoring type(s) applied.
2. Then, briefly justify each identified refactoring with reference to the specific code changes.
3. Only include refactorings from the list above.
4. Be concise but precise in your explanations.

Do not generate explanations unrelated to the given transformation.



- Extract Superclass
- Pull Up Field

## Justifications:

- **Extract Superclass:**
  - A new class, `EAddressResolverGroupProvider`, was created (new file `src/main/java/io/lettuce/core/resource/EAddressResolverGroupProvider.java`).
  - The existing class `AddressResolverGroupProvider` was modified to extend this new `EAddressResolverGroupProvider` class (its declaration changed from `class AddressResolverGroupProvider extends EAddressResolverGroupProvider`).
  - This refactoring was done to create a new level of abstraction and move common members to it.
- **Pull Up Field:**
  - The `static final` field `logger` was moved from the class `AddressResolverGroupProvider` to its new superclass `EAddressResolverGroupProvider`.
  - Specifically, the field `private static final InternalLogger logger = InternalLoggerFactory.getInstance(AddressResolverGroupProvider.class);` was removed from `AddressResolverGroupProvider.java`.
  - A corresponding field `protected static final InternalLogger logger = InternalLoggerFactory.getInstance(AddressResolverGroupProvider.class);` was added to the new superclass `EAddressResolverGroupProvider.java`. The visibility was changed from `private` to `protected` to allow access within the inheritance hierarchy if needed (though for static fields, access is typically by class name).
  - This moves a field to a superclass, often done as part of an "Extract Superclass" refactoring to share common attributes.

You are an expert coding assistant specialized in software refactoring, with many years of experience analyzing code transformations.

You will be given the diffs of a commit:

```
- **Diffs:**  
diff --git  
src/main/java/io/lettuce/core/dynamic/CommandMethod.java  
src/main/java/io/lettuce/core/dynamic/CommandMethod.java  
index 8d6e21bb..a73372b6 100644  
----  
src/main/java/io/lettuce/core/dynamic/CommandMethod.java  
+++  
src/main/java/io/lettuce/core/dynamic/CommandMethod.java  
@@ -15,7 +15,7 @@ import  
io.lettuce.core.dynamic.support.ResolvableType;  
 * @author Mark Paluch  
 * @since 5.0  
 */  
-public interface CommandMethod {  
+public interface CommandMethod extends  
ICommandMethod {  
  
    /**  
     * @return the method {@link Parameters}.  
@@ -61,12 +61,6 @@ public interface CommandMethod {  
    */  
    boolean isFutureExecution();  
  
-    /**  
-     * @return {@code true} if the method uses reactive  
-     execution declaring {@link org.reactivestreams.Publisher} as  
-     result  
-     *      type.  
-     */  
-    boolean isReactiveExecution();  
-  
    /**
```

```
* @return {@code true} if the method defines a {@link  
io.lettuce.core.dynamic.batch.CommandBatching} argument.  
 */  
  
diff --git  
src/main/java/io/lettuce/core/dynamic/ICommandMethod.java  
src/main/java/io/lettuce/core/dynamic/ICommandMethod.java  
new file mode 100644  
index 0000000..842c1e3e  
--- /dev/null  
+++  
src/main/java/io/lettuce/core/dynamic/ICommandMethod.java  
@@ -0,0 +1,11 @@  
+package io.lettuce.core.dynamic;  
  
+  
+public interface ICommandMethod {  
  
+  
+    /**  
+     * @return {@code true} if the method uses reactive  
execution declaring {@link org.reactivestreams.Publisher} as  
result  
+     * type.  
+    */  
+    boolean isReactiveExecution();  
  
+  
+}
```

Your task is to identify which refactoring type(s) have been applied in transforming the original program into the new version. Use only the following list of predefined refactorings:

(Add Method Parameter) – Introduces a new parameter to an existing method.

(Remove Method Parameter) – Eliminates an existing parameter from a method signature.

(Rename Method) – Changes the name of a method while preserving its behavior.

(Rename Class) – Changes the name of a class without altering its structure.

(Rename Package) – Changes the name of a package declaration.

(Rename Field) – Changes the name of a class or instance variable.

(Extract Class) – Moves a group of related fields and methods from an existing class into a newly created class.

(Extract Superclass) – Creates a new superclass to encapsulate shared attributes and behavior from two or more existing classes.

(Inline Method) – Replaces a method call with the method's body.

(Pull Up Method) – Pull a method from a child class (subclass) in the original code to its parent class (superclass) in the transformed version.

(Push Down Method) – Push a method from a parent class (superclass) in the original code to one or more of its child classes (subclasses) in the transformed version.

(Pull Up Field) – Pull a field from a child class (subclass) in the original code to its parent class (superclass) in the transformed version.

(Push Down Field) – Push a field from a parent class (superclass) in the original code to one or more of its child classes (subclasses) in the transformed version.

(Inline Class) – Merges a class into another when it is too small or redundant.

(Extract Interface) – Creates a new interface from existing method(s) in a class.

(Move Method) – Relocates a method to a more appropriate class.

(Move Field) – Relocates a field to a more appropriate class.

(Replace Magic Number with Constant) – Replaces a literal number with a named constant.

(Encapsulate Field) – Change a field access from not private to private and create get and set methods.

**\*\*Instructions:\*\***

1. Begin your response with a bullet-point list of the refactoring type(s) applied.
2. Then, briefly justify each identified refactoring with reference to the specific code changes.
3. Only include refactorings from the list above.
4. Be concise but precise in your explanations.

Do not generate explanations unrelated to the given transformation.



⋮

- Extract Superclass
- Pull Up Method

#### Justifications:

- **Extract Superclass:**
  - A new interface, `ICommandMethod`, was created (new file `src/main/java/io/lettuce/core/dynamic/ICommandMethod.java`). This new interface acts as a "superclass" or, more precisely, a "super-interface" in the type hierarchy.
  - The existing interface `CommandMethod` was modified to extend this new `ICommandMethod` interface (its declaration changed from `public interface CommandMethod` to `public interface CommandMethod extends ICommandMethod`).
  - This refactoring was performed to create a more generalized parent type for `CommandMethod` and move common method declarations to it. While the term "Extract Superclass" typically refers to classes, the structural pattern of creating a new parent type in an inheritance/extension hierarchy and moving members to it applies here to interfaces as well.
- **Pull Up Method:**
  - The method declaration `boolean isReactiveExecution()` was moved from the interface `CommandMethod` to its new super-interface `ICommandMethod`.

- Specifically, the declaration of `isReactiveExecution()` was removed from `CommandMethod.java`.
- An identical method declaration was added to the new super-interface `ICommandMethod.java`.
- This moves a method declaration up the inheritance hierarchy from a "sub-interface" to a "super-interface", which is the essence of the "Pull Up Method" refactoring.

You are an expert coding assistant specialized in software refactoring, with many years of experience analyzing code transformations.

You will be given the diffs of a commit:

```
- **Diffs:**  
diff --git  
src/main/java/io/lettuce/core/dynamic/DefaultRedisCommands  
Metadata.java  
src/main/java/io/lettuce/core/dynamic/DefaultRedisCommands  
Metadata.java  
index 759ae76b..9e7d15ab 100644  
---  
src/main/java/io/lettuce/core/dynamic/DefaultRedisCommands  
Metadata.java  
+++  
src/main/java/io/lettuce/core/dynamic/DefaultRedisCommands  
Metadata.java  
@@ -32,7 +32,6 @@ class DefaultRedisCommandsMetadata  
implements RedisCommandsMetadata {  
    this.apilInterface = apilInterface;  
}  
  
-  @Override  
    public Class<?> getCommandsInterface() {
```

```
        return apiInterface;
    }

diff --git
src/main/java/io/lettuce/core/dynamic/RedisCommandsMetadata.java
src/main/java/io/lettuce/core/dynamic/RedisCommandsMetadata.java
index 9201cb51..f9736b1d 100644
-----
src/main/java/io/lettuce/core/dynamic/RedisCommandsMetadata.java
+++  
src/main/java/io/lettuce/core/dynamic/RedisCommandsMetadata.java
@@ -14,13 +14,6 @@ interface RedisCommandsMetadata {
```

Collection<Method> getMethods();

```
- /**
- * Returns the Redis Commands interface.
- *
- * @return
- */
- Class<?> getCommandsInterface();
-
/**  
 * Lookup an interface annotation.  
 *
```

Your task is to identify which refactoring type(s) have been applied in transforming the original program into the new version. Use only the following list of predefined refactorings:

(Add Method Parameter) – Introduces a new parameter to an existing method.

- (Remove Method Parameter) – Eliminates an existing parameter from a method signature.
- (Rename Method) – Changes the name of a method while preserving its behavior.
- (Rename Class) – Changes the name of a class without altering its structure.
- (Rename Package) – Changes the name of a package declaration.
- (Rename Field) – Changes the name of a class or instance variable.
- (Extract Class) – Moves a group of related fields and methods from an existing class into a newly created class.
- (Extract Superclass) – Creates a new superclass to encapsulate shared attributes and behavior from two or more existing classes.
- (Inline Method) – Replaces a method call with the method's body.
- (Pull Up Method) – Pull a method from a child class (subclass) in the original code to its parent class (superclass) in the transformed version.
- (Push Down Method) – Push a method from a parent class (superclass) in the original code to one or more of its child classes (subclasses) in the transformed version.
- (Pull Up Field) – Pull a field from a child class (subclass) in the original code to its parent class (superclass) in the transformed version.
- (Push Down Field) – Push a field from a parent class (superclass) in the original code to one or more of its child classes (subclasses) in the transformed version.
- (Inline Class) – Merges a class into another when it is too small or redundant.
- (Extract Interface) – Creates a new interface from existing method(s) in a class.
- (Move Method) – Relocates a method to a more appropriate class.
- (Move Field) – Relocates a field to a more appropriate class.

(Replace Magic Number with Constant) – Replaces a literal number with a named constant.

(Encapsulate Field) – Change a field access from not private to private and create get and set methods.

**\*\*Instructions:\*\***

1. Begin your response with a bullet-point list of the refactoring type(s) applied.
2. Then, briefly justify each identified refactoring with reference to the specific code changes.
3. Only include refactorings from the list above.
4. Be concise but precise in your explanations.

Do not generate explanations unrelated to the given transformation.



- Push Down Method

#### Justifications:

- Push Down Method:

- The method declaration `Class<?> getCommandsInterface()` was removed from the `RedisCommandsMetadata` interface.
- The class `DefaultRedisCommandsMetadata`, which implements `RedisCommandsMetadata`, retained its implementation of the `getCommandsInterface()` method.
- The `@Override` annotation was removed from the `getCommandsInterface()` method in `DefaultRedisCommandsMetadata`. This signifies that the method is no longer an implementation of a method declared in the `RedisCommandsMetadata` interface.
- This change effectively moves the responsibility for this method from the general interface contract to the specific implementing class (or classes, if others also retained/added it). The method is "pushed down" from the abstraction (interface) to the concrete implementation.

You are an expert coding assistant specialized in software refactoring, with many years of experience analyzing code transformations.

You will be given the diffs of a commit:

```
- **Diffs:**  
diff --git  
src/main/java/io/lettuce/core/output/CommandOutput.java  
src/main/java/io/lettuce/core/output/CommandOutput.java  
index 5775f896..960106ae 100644  
--- src/main/java/io/lettuce/core/output/CommandOutput.java  
+++ src/main/java/io/lettuce/core/output/CommandOutput.java  
@@ -50,6 +50,8 @@ public abstract class  
CommandOutput<K, V, T> {  
  
protected String error;  
  
+ protected boolean initialized;  
+  
/**  
 * Initialize a new instance that encodes and decodes keys  
and values using the supplied codec.  
*  
diff --git  
src/main/java/io/lettuce/core/output/DoubleListOutput.java  
src/main/java/io/lettuce/core/output/DoubleListOutput.java  
index 72a2e5a4..28228a86 100644  
--- src/main/java/io/lettuce/core/output/DoubleListOutput.java  
+++  
src/main/java/io/lettuce/core/output/DoubleListOutput.java  
@@ -38,8 +38,6 @@ import  
io.lettuce.core.codec.RedisCodec;  
*/
```

```
public class DoubleListOutput<K, V> extends  
CommandOutput<K, V, List<Double>> {  
  
    - private boolean initialized;  
  
    -  
  
        public DoubleListOutput(RedisCodec<K, V> codec) {  
            super(codec, new ArrayList<>());  
        }  
}
```

Your task is to identify which refactoring type(s) have been applied in transforming the original program into the new version. Use only the following list of predefined refactorings:

(Add Method Parameter) – Introduces a new parameter to an existing method.

(Remove Method Parameter) – Eliminates an existing parameter from a method signature.

(Rename Method) – Changes the name of a method while preserving its behavior.

(Rename Class) – Changes the name of a class without altering its structure.

(Rename Package) – Changes the name of a package declaration.

(Rename Field) – Changes the name of a class or instance variable.

(Extract Class) – Moves a group of related fields and methods from an existing class into a newly created class.

(Extract Superclass) – Creates a new superclass to encapsulate shared attributes and behavior from two or more existing classes.

(Inline Method) – Replaces a method call with the method's body.

(Pull Up Method) – Pull a method from a child class (subclass) in the original code to its parent class (superclass) in the transformed version.

(Push Down Method) – Push a method from a parent class (superclass) in the original code to one or more of its child classes (subclasses) in the transformed version.

(Pull Up Field) – Pull a field from a child class (subclass) in the original code to its parent class (superclass) in the transformed version.

(Push Down Field) – Push a field from a parent class (superclass) in the original code to one or more of its child classes (subclasses) in the transformed version.

(Inline Class) – Merges a class into another when it is too small or redundant.

(Extract Interface) – Creates a new interface from existing method(s) in a class.

(Move Method) – Relocates a method to a more appropriate class.

(Move Field) – Relocates a field to a more appropriate class.

(Replace Magic Number with Constant) – Replaces a literal number with a named constant.

(Encapsulate Field) – Change a field access from not private to private and create get and set methods.

**\*\*Instructions:\*\***

1. Begin your response with a bullet-point list of the refactoring type(s) applied.
2. Then, briefly justify each identified refactoring with reference to the specific code changes.
3. Only include refactorings from the list above.
4. Be concise but precise in your explanations.

Do not generate explanations unrelated to the given transformation.



:

- Pull Up Field

**Justifications:**

- **Pull Up Field:**

- The field `initialized` of type `boolean` was moved from the subclass `DoubleListOutput` to its superclass `CommandOutput`.
- In `DoubleListOutput.java` (the subclass), the field `private boolean initialized;` was removed.
- In `CommandOutput.java` (the superclass), a new field `protected boolean initialized;` was added.
- This refactoring is used to move a common field from one or more subclasses to their shared superclass, reducing duplication and centralizing the attribute. The visibility was changed from `private` to `protected` to allow subclasses to access the field inherited from the superclass.

You are an expert coding assistant specialized in software refactoring, with many years of experience analyzing code transformations.

You will be given the diffs of a commit:

```
- **Diffs:**  
diff --git  
src/main/java/io/lettuce/core/output/IScoredValueStreamingCh  
annel.java  
src/main/java/io/lettuce/core/output/IScoredValueStreamingCh  
annel.java  
new file mode 100644  
index 0000000..ffcac075  
--- /dev/null  
+++  
src/main/java/io/lettuce/core/output/IScoredValueStreamingCh  
annel.java  
@@ -0,0 +1,14 @@  
+package io.lettuce.core.output;  
+  
+import io.lettuce.core.ScoredValue;
```

```
+  
+public interface IScoredValueStreamingChannel<V> {  
+  
+    /**  
+     * Called on every incoming ScoredValue.  
+     *  
+     * @param value the scored value  
+     */  
+    void onValue(ScoredValue<V> value);  
+  
+}  
diff --git  
src/main/java/io/lettuce/core/output/ScoredValueStreamingCh  
annel.java  
src/main/java/io/lettuce/core/output/ScoredValueStreamingCh  
annel.java  
index 6b8edbbf..ab852438 100644  
---  
src/main/java/io/lettuce/core/output/ScoredValueStreamingCh  
annel.java  
+++  
src/main/java/io/lettuce/core/output/ScoredValueStreamingCh  
annel.java  
@@ -11,13 +11,6 @@ import io.lettuce.core.ScoredValue;  
 * @since 3.0  
 */  
@FunctionalInterface  
-public interface ScoredValueStreamingChannel<V> extends  
StreamingChannel {  
-  
-    /**  
-     * Called on every incoming ScoredValue.  
-     *  
-     * @param value the scored value  
-     */  
-    void onValue(ScoredValue<V> value);
```

```
+public interface ScoredValueStreamingChannel<V> extends  
StreamingChannel, IScoredValueStreamingChannel<V> {  
  
}
```

Your task is to identify which refactoring type(s) have been applied in transforming the original program into the new version. Use only the following list of predefined refactorings:

(Add Method Parameter) – Introduces a new parameter to an existing method.

(Remove Method Parameter) – Eliminates an existing parameter from a method signature.

(Rename Method) – Changes the name of a method while preserving its behavior.

(Rename Class) – Changes the name of a class without altering its structure.

(Rename Package) – Changes the name of a package declaration.

(Rename Field) – Changes the name of a class or instance variable.

(Extract Class) – Moves a group of related fields and methods from an existing class into a newly created class.

(Extract Superclass) – Creates a new superclass to encapsulate shared attributes and behavior from two or more existing classes.

(Inline Method) – Replaces a method call with the method's body.

(Pull Up Method) – Pull a method from a child class (subclass) in the original code to its parent class (superclass) in the transformed version.

(Push Down Method) – Push a method from a parent class (superclass) in the original code to one or more of its child classes (subclasses) in the transformed version.

(Pull Up Field) – Pull a field from a child class (subclass) in the original code to its parent class (superclass) in the transformed version.

(Push Down Field) – Push a field from a parent class (superclass) in the original code to one or more of its child classes (subclasses) in the transformed version.

(Inline Class) – Merges a class into another when it is too small or redundant.

(Extract Interface) – Creates a new interface from existing method(s) in a class.

(Move Method) – Relocates a method to a more appropriate class.

(Move Field) – Relocates a field to a more appropriate class.

(Replace Magic Number with Constant) – Replaces a literal number with a named constant.

(Encapsulate Field) – Change a field access from not private to private and create get and set methods.

**\*\*Instructions:\*\***

1. Begin your response with a bullet-point list of the refactoring type(s) applied.
2. Then, briefly justify each identified refactoring with reference to the specific code changes.
3. Only include refactorings from the list above.
4. Be concise but precise in your explanations.

Do not generate explanations unrelated to the given transformation.



- Extract Superclass
- Pull Up Method

#### Justifications:

- Extract Superclass:

- A new interface, `IScoredValueStreamingChannel<V>`, was created (new file `src/main/java/io/lettuce/core/output/IScoredValueStreamingChannel.java`). This new interface acts as a "superclass" or, more accurately, a "super-interface" in the type hierarchy.
  - The existing interface `ScoredValueStreamingChannel<V>` was modified to extend this new `IScoredValueStreamingChannel<V>` interface (its declaration changed from `public interface ScoredValueStreamingChannel<V> extends StreamingChannel` to `public interface ScoredValueStreamingChannel<V> extends StreamingChannel, IScoredValueStreamingChannel<V>`).
  - This refactoring created a new, more specialized parent type to hold a specific part of the contract previously defined in `ScoredValueStreamingChannel`. While "Extract Superclass" usually pertains to classes, the structural change of creating a new parent in an inheritance hierarchy and moving members to it applies analogously to interfaces.
- **Pull Up Method:**
- The method declaration `void onValue(ScoredValue<V> value)` was moved from the interface `ScoredValueStreamingChannel<V>` to its new super-interface `IScoredValueStreamingChannel<V>`.
  - Specifically, the declaration of `onValue(ScoredValue<V> value)` was removed from the body of `ScoredValueStreamingChannel.java`.
  - An identical method declaration was added to the new super-interface `IScoredValueStreamingChannel.java`.
  - This action moves a method declaration up the inheritance hierarchy from a "sub-interface" to a "super-interface".

You are an expert coding assistant specialized in software refactoring, with many years of experience analyzing code transformations.

You will be given the diffs of a commit:

- \*\*Diffs:\*\*

diff --git

src/main/java/io/lettuce/core/output/StringMatchResultOutput.

```
java
src/main/java/io/lettuce/core/output/StringMatchResultOutput.

java
index 62c6f35a..ecb59cd0 100644
---

src/main/java/io/lettuce/core/output/StringMatchResultOutput.
java
+++
src/main/java/io/lettuce/core/output/StringMatchResultOutput.
java
@@ -78,7 +78,16 @@ public class StringMatchResultOutput<K,
V> extends CommandOutput<K, V, StringMat
    public void complete(int depth) {

        if (depth == 2) {
-
            matchedPositions.add(buildMatchedString(positions));
+
+           if (positions == null) {
+
+               throw new IllegalStateException("No matched
positions");
+
+           }
+
+           int size = positions.size();
+
+           // not WITHMATCHLEN
+
+           long matchLen = size % 2 == 0 ? OL :
positions.get(size - 1);
+
+           matchedPositions.add(new MatchedPosition(new
Position(positions.get(0), positions.get(1)),
+
+               new Position(positions.get(2), positions.get(3)),
matchLen));
+
+           positions = null;
}
}

@@ -87,17 +96,4 @@ public class StringMatchResultOutput<K,
V> extends CommandOutput<K, V, StringMat
```

```
        }
    }

    - private static MatchedPosition
    buildMatchedString(List<Long> positions) {
    -
    -
    -     if (positions == null) {
    -         throw new IllegalStateException("No matched
    positions");
    -     }
    -
    -
    -     int size = positions.size();
    -     // not WITHMATCHLEN
    -     long matchLen = size % 2 == 0 ? 0L : positions.get(size -
    1);
    -     return new MatchedPosition(new
    Position(positions.get(0), positions.get(1)),
    -         new Position(positions.get(2), positions.get(3)),
    matchLen);
    - }
    -
    }
```

Your task is to identify which refactoring type(s) have been applied in transforming the original program into the new version. Use only the following list of predefined refactorings:

(Add Method Parameter) – Introduces a new parameter to an existing method.

(Remove Method Parameter) – Eliminates an existing parameter from a method signature.

(Rename Method) – Changes the name of a method while preserving its behavior.

(Rename Class) – Changes the name of a class without altering its structure.

(Rename Package) – Changes the name of a package declaration.

(Rename Field) – Changes the name of a class or instance variable.

(Extract Class) – Moves a group of related fields and methods from an existing class into a newly created class.

(Extract Superclass) – Creates a new superclass to encapsulate shared attributes and behavior from two or more existing classes.

(Inline Method) – Replaces a method call with the method's body.

(Pull Up Method) – Pull a method from a child class (subclass) in the original code to its parent class (superclass) in the transformed version.

(Push Down Method) – Push a method from a parent class (superclass) in the original code to one or more of its child classes (subclasses) in the transformed version.

(Pull Up Field) – Pull a field from a child class (subclass) in the original code to its parent class (superclass) in the transformed version.

(Push Down Field) – Push a field from a parent class (superclass) in the original code to one or more of its child classes (subclasses) in the transformed version.

(Inline Class) – Merges a class into another when it is too small or redundant.

(Extract Interface) – Creates a new interface from existing method(s) in a class.

(Move Method) – Relocates a method to a more appropriate class.

(Move Field) – Relocates a field to a more appropriate class.

(Replace Magic Number with Constant) – Replaces a literal number with a named constant.

(Encapsulate Field) – Change a field access from not private to private and create get and set methods.

**\*\*Instructions:\*\***

1. Begin your response with a bullet-point list of the refactoring type(s) applied.
2. Then, briefly justify each identified refactoring with reference to the specific code changes.
3. Only include refactorings from the list above.
4. Be concise but precise in your explanations.

Do not generate explanations unrelated to the given transformation.



⋮

- **Inline Method**

#### Justifications:

- **Inline Method:**

- The private static method `buildMatchedString(List<Long> positions)` was removed from the `StringMatchResultOutput` class.
- The body of the `buildMatchedString` method was directly inserted into its call site within the `complete(int depth)` method.
- Specifically, the line `matchedPositions.add(buildMatchedString(positions));` in the `complete` method was replaced with the logic previously contained in `buildMatchedString`, using the `positions` field of the class.
- This refactoring replaces a method call with the method's actual code, which is useful when a method's body is simple or it's only used in one place, potentially improving readability or removing unnecessary indirection.

You are an expert coding assistant specialized in software refactoring, with many years of experience analyzing code transformations.

You will be given the diffs of a commit:

```
- ***Diffs:**  
diff --git  
src/main/java/io/lettuce/core/protocol/ConnectionWatchdog.ja  
va  
src/main/java/io/lettuce/core/protocol/ConnectionWatchdog.ja  
va  
index 84bcb41f..69c7c492 100644  
---  
src/main/java/io/lettuce/core/protocol/ConnectionWatchdog.ja  
va  
+++  
src/main/java/io/lettuce/core/protocol/ConnectionWatchdog.ja  
va  
@@ -21,6 +21,8 @@ package io.lettuce.core.protocol;  
  
import java.net.SocketAddress;  
import java.time.Duration;  
+import java.util.ArrayList;  
+import java.util.Collection;  
import java.util.concurrent.CompletableFuture;  
import java.util.concurrent.TimeUnit;  
import java.util.concurrent.atomic.AtomicBoolean;  
@@ -436,4 +438,20 @@ public class ConnectionWatchdog  
extends ChannelInboundHandlerAdapter {  
    return logPrefix = buffer;  
}  
  
+  <K, V> Collection<RedisCommand<K, V, ?>>  
processActivationCommands(Collection<? extends  
RedisCommand<K, V, ?>> commands) {  
+  
+    Collection<RedisCommand<K, V, ?>>  
commandsToReturn = new ArrayList<>(commands.size());  
+  
+    for (RedisCommand<K, V, ?> command : commands) {  
+
```

```

+     if
(!ActivationCommand.isActivationCommand(command)) {
+         command = new ActivationCommand<>(command);
+
+
+         commandsToReturn.add(command);
}
+
+
+     return commandsToReturn;
}
+
}

diff --git
src/main/java/io/lettuce/core/protocol/DefaultEndpoint.java
src/main/java/io/lettuce/core/protocol/DefaultEndpoint.java
index 79f2f05f..6f61a525 100644
--- src/main/java/io/lettuce/core/protocol/DefaultEndpoint.java
+++ src/main/java/io/lettuce/core/protocol/DefaultEndpoint.java
@@ -236,7 +236,7 @@ public class DefaultEndpoint
implements RedisChannelWriter, Endpoint, PushHandle
    sharedLock.incrementWriters();

if (inActivation) {
-
    commands =
processActivationCommands(commands);
+
    commands =
connectionWatchdog.processActivationCommands(command
s);
}

if (autoFlushCommands) {
@@ -269,23 +269,6 @@ public class DefaultEndpoint
implements RedisChannelWriter, Endpoint, PushHandle
    return command;
}

```

```

-    private <K, V> Collection<RedisCommand<K, V, ?>>
processActivationCommands(
-        Collection<? extends RedisCommand<K, V, ?>>
commands) {
-
-        Collection<RedisCommand<K, V, ?>> commandsToReturn
= new ArrayList<>(commands.size());
-
-        for (RedisCommand<K, V, ?> command : commands) {
-
-            if
(!ActivationCommand.isActivationCommand(command)) {
-                command = new ActivationCommand<>(command);
-            }
-
-            commandsToReturn.add(command);
-        }
-
-        return commandsToReturn;
-    }
-
private RedisException validateWrite(int commands) {
-
    if (isClosed()) {

```

Your task is to identify which refactoring type(s) have been applied in transforming the original program into the new version. Use only the following list of predefined refactorings:

(Add Method Parameter) – Introduces a new parameter to an existing method.

(Remove Method Parameter) – Eliminates an existing parameter from a method signature.

(Rename Method) – Changes the name of a method while preserving its behavior.

(Rename Class) – Changes the name of a class without altering its structure.

(Rename Package) – Changes the name of a package declaration.

(Rename Field) – Changes the name of a class or instance variable.

(Extract Class) – Moves a group of related fields and methods from an existing class into a newly created class.

(Extract Superclass) – Creates a new superclass to encapsulate shared attributes and behavior from two or more existing classes.

(Inline Method) – Replaces a method call with the method's body.

(Pull Up Method) – Pull a method from a child class (subclass) in the original code to its parent class (superclass) in the transformed version.

(Push Down Method) – Push a method from a parent class (superclass) in the original code to one or more of its child classes (subclasses) in the transformed version.

(Pull Up Field) – Pull a field from a child class (subclass) in the original code to its parent class (superclass) in the transformed version.

(Push Down Field) – Push a field from a parent class (superclass) in the original code to one or more of its child classes (subclasses) in the transformed version.

(Inline Class) – Merges a class into another when it is too small or redundant.

(Extract Interface) – Creates a new interface from existing method(s) in a class.

(Move Method) – Relocates a method to a more appropriate class.

(Move Field) – Relocates a field to a more appropriate class.

(Replace Magic Number with Constant) – Replaces a literal number with a named constant.

(Encapsulate Field) – Change a field access from not private to private and create get and set methods.

**\*\*Instructions:\*\***

1. Begin your response with a bullet-point list of the refactoring type(s) applied.
2. Then, briefly justify each identified refactoring with reference to the specific code changes.
3. Only include refactorings from the list above.
4. Be concise but precise in your explanations.

Do not generate explanations unrelated to the given transformation.



⋮

- Move Method

#### Justifications:

- **Move Method:**

- The method `processActivationCommands(Collection<? extends RedisCommand<K, V, ?>> commands)` was moved from the `DefaultEndpoint` class to the `ConnectionWatchdog` class.
- In `DefaultEndpoint.java`, the `private` method `processActivationCommands` was removed.
- An identical method (same logic and parameters, though visibility changed from `private` to package-private) was added to `ConnectionWatchdog.java`.
- The call to this method in `DefaultEndpoint.java` was updated from a local call `processActivationCommands(commands)` to `connectionWatchdog.processActivationCommands(commands)`, indicating the method is now invoked on an instance of `ConnectionWatchdog`.
- This refactoring relocates a method to a class where it might be more appropriate or to improve cohesion/coupling between classes.

You are an expert coding assistant specialized in software refactoring, with many years of experience analyzing code transformations.

You will be given the diffs of a commit:

```
- **Diffs:**  
diff --git  
src/main/java/io/lettuce/core/protocol/CommandHandler.java  
src/main/java/io/lettuce/core/protocol/CommandHandler.java  
index 59aee61e..378c0b01 100644  
---  
src/main/java/io/lettuce/core/protocol/CommandHandler.java  
+++  
src/main/java/io/lettuce/core/protocol/CommandHandler.java  
@@ -575,7 +575,7 @@ public class CommandHandler extends  
ChannelDuplexHandler implements HasQueuedCom  
    return command;  
}  
  
-   LatencyMeteredCommand<?, ?, ?>  
latencyMeteredCommand = new LatencyMeteredCommand<>  
(command);  
+   RLatencyMeteredCommand<?, ?, ?>  
latencyMeteredCommand = new  
RLatencyMeteredCommand<>(command);  
    latencyMeteredCommand.firstResponse(-1);  
    latencyMeteredCommand.sent(nanoTime());  
  
diff --git  
src/main/java/io/lettuce/core/protocol/LatencyMeteredComma  
nd.java  
src/main/java/io/lettuce/core/protocol/RLatencyMeteredComm  
and.java  
similarity index 84%
```

```
rename from
src/main/java/io/lettuce/core/protocol/LatencyMeteredComma
nd.java
rename to
src/main/java/io/lettuce/core/protocol/RLatencyMeteredComm
and.java
index 26e7fb0e..bd759f3b 100644
---
src/main/java/io/lettuce/core/protocol/LatencyMeteredComma
nd.java
++++
src/main/java/io/lettuce/core/protocol/RLatencyMeteredComm
and.java
@@ -6,7 +6,7 @@ package io.lettuce.core.protocol;
 * @author Mark Paluch
 * @since 4.4
 */
-class LatencyMeteredCommand<K, V, T> extends
CommandWrapper<K, V, T> implements WithLatency {
+class RLatencyMeteredCommand<K, V, T> extends
CommandWrapper<K, V, T> implements WithLatency {

    private long sentNs = -1;

@@ -14,7 +14,7 @@ class LatencyMeteredCommand<K, V, T>
extends CommandWrapper<K, V, T> implements

    private long completedNs = -1;

-    public LatencyMeteredCommand(RedisCommand<K, V, T>
command) {
+    public RLatencyMeteredCommand(RedisCommand<K, V,
T> command) {
        super(command);
    }
}
```

```
diff --git
src/test/java/io/lettuce/core/protocol/CommandHandlerUnitTe
sts.java
src/test/java/io/lettuce/core/protocol/CommandHandlerUnitTe
sts.java
index e7f7cfec..88039d98 100644
-----
src/test/java/io/lettuce/core/protocol/CommandHandlerUnitTe
sts.java
+++  

src/test/java/io/lettuce/core/protocol/CommandHandlerUnitTe
sts.java
@@ -367,7 +367,7 @@ class CommandHandlerUnitTests {
    sut.write(context, command, promise);

    verify(context).write(command, promise);
-   assertThat(stack).hasSize(1).allMatch(o -> o instanceof
LatencyMeteredCommand);
+   assertThat(stack).hasSize(1).allMatch(o -> o instanceof
RLatencyMeteredCommand);
}

@Test
@@ -463,7 +463,7 @@ class CommandHandlerUnitTests {
    verify(context).write(captor.capture(), any());

    assertThat(captor.getValue()).containsOnly(command2);
-   assertThat(stack).hasSize(1).allMatch(o -> o instanceof
LatencyMeteredCommand)
+   assertThat(stack).hasSize(1).allMatch(o -> o instanceof
RLatencyMeteredCommand)
.allMatch(o ->
CommandWrapper.unwrap((RedisCommand) o) ==
command2);
}
```

```
@@ -481,7 +481,7 @@ class CommandHandlerUnitTests {
```

```
    sut.channelRead(context,  
    Unpooled.wrappedBuffer("*1\r\n+OK\r\n".getBytes()));
```

```
-    verify(latencyCollector).recordCommandLatency(any(),  
any(), any(LatencyMeteredCommand.class), gt(OL), gt(OL));  
+    verify(latencyCollector).recordCommandLatency(any(),  
any(), any(RLatencyMeteredCommand.class), gt(OL), gt(OL));
```

```
    sut.channelUnregistered(context);  
}
```

```
diff --git  
src/test/java/io/lettuce/core/protocol/DefaultEndpointUnitTest  
s.java  
src/test/java/io/lettuce/core/protocol/DefaultEndpointUnitTest  
s.java  
index 06354e5d..c42951c8 100644  
---
```

```
src/test/java/io/lettuce/core/protocol/DefaultEndpointUnitTest  
s.java  
+++  
src/test/java/io/lettuce/core/protocol/DefaultEndpointUnitTest  
s.java
```

```
@@ -413,7 +413,7 @@ class DefaultEndpointUnitTests {  
    when(channel.isActive()).thenReturn(true);  
    ConnectionTestUtil.getDisconnectedBuffer(sut)  
        .add(new ActivationCommand<>(new Command<>  
(CommandType.SELECT, new StatusOutput<>  
(StringCodec.UTF8))));
```

```
-  
ConnectionTestUtil.getDisconnectedBuffer(sut).add(new  
LatencyMeteredCommand<>(  
+  
ConnectionTestUtil.getDisconnectedBuffer(sut).add(new  
RLatencyMeteredCommand<>(
```

```
    new ActivationCommand<>(new Command<>  
(CommandType.SUBSCRIBE, new StatusOutput<>  
(StringCodec.UTF8)))));
```

```
    doAnswer(i -> {
```

Your task is to identify which refactoring type(s) have been applied in transforming the original program into the new version. Use only the following list of predefined refactorings:

(Add Method Parameter) – Introduces a new parameter to an existing method.

(Remove Method Parameter) – Eliminates an existing parameter from a method signature.

(Rename Method) – Changes the name of a method while preserving its behavior.

(Rename Class) – Changes the name of a class without altering its structure.

(Rename Package) – Changes the name of a package declaration.

(Rename Field) – Changes the name of a class or instance variable.

(Extract Class) – Moves a group of related fields and methods from an existing class into a newly created class.

(Extract Superclass) – Creates a new superclass to encapsulate shared attributes and behavior from two or more existing classes.

(Inline Method) – Replaces a method call with the method's body.

(Pull Up Method) – Pull a method from a child class (subclass) in the original code to its parent class (superclass) in the transformed version.

(Push Down Method) – Push a method from a parent class (superclass) in the original code to one or more of its child classes (subclasses) in the transformed version.

(Pull Up Field) – Pull a field from a child class (subclass) in the original code to its parent class (superclass) in the transformed version.

(Push Down Field) – Push a field from a parent class (superclass) in the original code to one or more of its child classes (subclasses) in the transformed version.

(Inline Class) – Merges a class into another when it is too small or redundant.

(Extract Interface) – Creates a new interface from existing method(s) in a class.

(Move Method) – Relocates a method to a more appropriate class.

(Move Field) – Relocates a field to a more appropriate class.

(Replace Magic Number with Constant) – Replaces a literal number with a named constant.

(Encapsulate Field) – Change a field access from not private to private and create get and set methods.

**\*\*Instructions:\*\***

1. Begin your response with a bullet-point list of the refactoring type(s) applied.
2. Then, briefly justify each identified refactoring with reference to the specific code changes.
3. Only include refactorings from the list above.
4. Be concise but precise in your explanations.

Do not generate explanations unrelated to the given transformation.



⋮

- Rename Class

#### Justifications:

- **Rename Class:**

- The class `LatencyMeteredCommand` was renamed to `RLatencyMeteredCommand`.

- This is indicated by the file rename operation: `rename from src/main/java/io/lettuce/core/protocol/LatencyMeteredCommand.java rename to src/main/java/io/lettuce/core/protocol/RLatencyMeteredCommand.java`.
- The class declaration itself changed from `class LatencyMeteredCommand<K, V, T>` to `class RLatencyMeteredCommand<K, V, T>`.
- Consequently, the constructor name changed from `LatencyMeteredCommand` to `RLatencyMeteredCommand`.
- All usages of the class, including instantiations (e.g., `new LatencyMeteredCommand<>(...)`) to `new RLatencyMeteredCommand<>(...)` in `CommandHandler.java` and `DefaultEndpointUnitTests.java`) and type checks (e.g., `instanceof LatencyMeteredCommand` to `instanceof RLatencyMeteredCommand` in `CommandHandlerUnitTests.java`), were updated to reflect the new class name.

You are an expert coding assistant specialized in software refactoring, with many years of experience analyzing code transformations.

You will be given the diffs of a commit:

```
- **Diffs:**  
diff --git  
src/main/java/io/lettuce/core/resource/AddressResolverGroup  
Provider.java  
src/main/java/io/lettuce/core/resource/AddressResolverGroup  
Provider.java  
index 1b59d949..e95d8544 100644  
---  
src/main/java/io/lettuce/core/resource/AddressResolverGroup  
Provider.java  
+++  
src/main/java/io/lettuce/core/resource/AddressResolverGroup  
Provider.java  
@@ -22,6 +22,8 @@ import  
io.netty.util.internal.logging.InternalLoggerFactory;
```

```
 */
class AddressResolverGroupProvider extends
EAddressResolverGroupProvider {

    + protected static final InternalLogger logger =
InternalLoggerFactory.getInstance(AddressResolverGroupPro
vider.class);

    +
    private static final AddressResolverGroup<?>
ADDRESS_RESOLVER_GROUP;

    static {
diff --git
src/main/java/io/lettuce/core/resource/EAddressResolverGrou
pProvider.java
src/main/java/io/lettuce/core/resource/EAddressResolverGrou
pProvider.java
index Obbf5b2f..9f6e988e 100644
---
src/main/java/io/lettuce/core/resource/EAddressResolverGrou
pProvider.java
++++
src/main/java/io/lettuce/core/resource/EAddressResolverGrou
pProvider.java
@@ -1,10 +1,5 @@
package io.lettuce.core.resource;

-import io.netty.util.internal.logging.InternalLogger;
-import io.netty.util.internal.logging.InternalLoggerFactory;
-
public class EAddressResolverGroupProvider {

    - protected static final InternalLogger logger =
InternalLoggerFactory.getInstance(AddressResolverGroupPro
vider.class);
-
```

```
}
```

Your task is to identify which refactoring type(s) have been applied in transforming the original program into the new version. Use only the following list of predefined refactorings:

(Add Method Parameter) – Introduces a new parameter to an existing method.

(Remove Method Parameter) – Eliminates an existing parameter from a method signature.

(Rename Method) – Changes the name of a method while preserving its behavior.

(Rename Class) – Changes the name of a class without altering its structure.

(Rename Package) – Changes the name of a package declaration.

(Rename Field) – Changes the name of a class or instance variable.

(Extract Class) – Moves a group of related fields and methods from an existing class into a newly created class.

(Extract Superclass) – Creates a new superclass to encapsulate shared attributes and behavior from two or more existing classes.

(Inline Method) – Replaces a method call with the method's body.

(Pull Up Method) – Pull a method from a child class (subclass) in the original code to its parent class (superclass) in the transformed version.

(Push Down Method) – Push a method from a parent class (superclass) in the original code to one or more of its child classes (subclasses) in the transformed version.

(Pull Up Field) – Pull a field from a child class (subclass) in the original code to its parent class (superclass) in the transformed version.

(Push Down Field) – Push a field from a parent class (superclass) in the original code to one or more of its child classes (subclasses) in the transformed version.

(Inline Class) – Merges a class into another when it is too small or redundant.

(Extract Interface) – Creates a new interface from existing method(s) in a class.

(Move Method) – Relocates a method to a more appropriate class.

(Move Field) – Relocates a field to a more appropriate class.

(Replace Magic Number with Constant) – Replaces a literal number with a named constant.

(Encapsulate Field) – Change a field access from not private to private and create get and set methods.

**\*\*Instructions:\*\***

1. Begin your response with a bullet-point list of the refactoring type(s) applied.
2. Then, briefly justify each identified refactoring with reference to the specific code changes.
3. Only include refactorings from the list above.
4. Be concise but precise in your explanations.

Do not generate explanations unrelated to the given transformation.



⋮

- Push Down Field

#### Justifications:

- Push Down Field:

- The `protected static final` field `logger` was moved from the superclass `EAddressResolverGroupProvider` to its subclass `AddressResolverGroupProvider`.
- In `EAddressResolverGroupProvider.java` (the superclass), the field `protected static final InternalLogger logger =`

```
InternalLoggerFactory.getInstance(AddressResolverGroupProvider.class);  
(and its associated imports) was removed.
```

- An identical field (same name, type, visibility, and initialization) was added to `AddressResolverGroupProvider.java` (the subclass).
- This refactoring moves a field from a superclass to one or more subclasses, typically when the field is only used by those specific subclasses.

You are an expert coding assistant specialized in software refactoring, with many years of experience analyzing code transformations.

You will be given the diffs of a commit:

```
- **Diffs:**  
diff --git  
src/main/java/io/lettuce/core/resource/EEpollProvider.java  
src/main/java/io/lettuce/core/resource/EEpollProvider.java  
new file mode 100644  
index 0000000..d980efb7  
--- /dev/null  
+++ src/main/java/io/lettuce/core/resource/EEpollProvider.java  
@@ -0,0 +1,15 @@  
+package io.lettuce.core.resource;  
+  
+import io.netty.bootstrap.Bootstrap;  
+import io.netty.channel.epoll.EpollChannelOption;  
+  
+import java.time.Duration;  
+  
+public class EEpollProvider {  
+    /**  
+     * Apply TcpUserTimeout options.  
+     */
```

```
+  public static void applyTcpUserTimeout(Bootstrap
bootstrap, Duration timeout) {
+
bootstrap.option(EpollChannelOption.TCP_USER_TIMEOUT,
Math.toIntExact(timeout.toMillis()));
+
+}
diff --git
src/main/java/io/lettuce/core/resource/EpollProvider.java
src/main/java/io/lettuce/core/resource/EpollProvider.java
index bb748334..a78f1508 100644
--- src/main/java/io/lettuce/core/resource/EpollProvider.java
+++ src/main/java/io/lettuce/core/resource/EpollProvider.java
@@ -48,7 +48,7 @@ import
io.netty.util.internal.logging.InternalLoggerFactory;
 * @author Yohei Ueki
 * @since 4.4
 */
-public class EpollProvider {
+public class EpollProvider extends EEpollProvider {

    private static final InternalLogger logger =
InternalLoggerFactory.getInstance(EpollProvider.class);

@@ -126,13 +126,6 @@ public class EpollProvider {
    bootstrap.option(EpollChannelOption.TCP_KEEPINTVL,
Math.toIntExact(interval.getSeconds()));
}

-
- /**
- * Apply TcpUserTimeout options.
- */
- public static void applyTcpUserTimeout(Bootstrap
bootstrap, Duration timeout) {
-
bootstrap.option(EpollChannelOption.TCP_USER_TIMEOUT,
```

```
Math.toIntExact(timeout.toMillis())));
-  }
-
/***
 * {@link EventLoopResources} for available Epoll.
*/

```

Your task is to identify which refactoring type(s) have been applied in transforming the original program into the new version. Use only the following list of predefined refactorings:

(Add Method Parameter) – Introduces a new parameter to an existing method.

(Remove Method Parameter) – Eliminates an existing parameter from a method signature.

(Rename Method) – Changes the name of a method while preserving its behavior.

(Rename Class) – Changes the name of a class without altering its structure.

(Rename Package) – Changes the name of a package declaration.

(Rename Field) – Changes the name of a class or instance variable.

(Extract Class) – Moves a group of related fields and methods from an existing class into a newly created class.

(Extract Superclass) – Creates a new superclass to encapsulate shared attributes and behavior from two or more existing classes.

(Inline Method) – Replaces a method call with the method's body.

(Pull Up Method) – Pull a method from a child class (subclass) in the original code to its parent class (superclass) in the transformed version.

(Push Down Method) – Push a method from a parent class (superclass) in the original code to one or more of its child classes (subclasses) in the transformed version.

(Pull Up Field) – Pull a field from a child class (subclass) in the original code to its parent class (superclass) in the transformed version.

(Push Down Field) – Push a field from a parent class (superclass) in the original code to one or more of its child classes (subclasses) in the transformed version.

(Inline Class) – Merges a class into another when it is too small or redundant.

(Extract Interface) – Creates a new interface from existing method(s) in a class.

(Move Method) – Relocates a method to a more appropriate class.

(Move Field) – Relocates a field to a more appropriate class.

(Replace Magic Number with Constant) – Replaces a literal number with a named constant.

(Encapsulate Field) – Change a field access from not private to private and create get and set methods.

**\*\*Instructions:\*\***

1. Begin your response with a bullet-point list of the refactoring type(s) applied.
2. Then, briefly justify each identified refactoring with reference to the specific code changes.
3. Only include refactorings from the list above.
4. Be concise but precise in your explanations.

Do not generate explanations unrelated to the given transformation.



- Extract Superclass
- Pull Up Method

#### Justifications:

- Extract Superclass:

- A new class, `EEpollProvider`, was created (new file `src/main/java/io/lettuce/core/resource/EEpollProvider.java`).
  - The existing class `EpollProvider` was modified to extend this new `EEpollProvider` class (its declaration changed from `public class EpollProvider` to `public class EpollProvider extends EEpollProvider`).
  - This refactoring was performed to create a new parent class to house common or related functionality.
- **Pull Up Method:**
- The `public static void applyTcpUserTimeout(Bootstrap bootstrap, Duration timeout)` method was moved from the `EpollProvider` class to its new superclass `EEpollProvider`.
  - Specifically, the `applyTcpUserTimeout` method was removed from `EpollProvider.java`.
  - An identical method (same signature, body, and static modifier) was added to the new superclass `EEpollProvider.java`.
  - This action moves a method from a subclass to its superclass, a common step in generalizing behavior or sharing it among potential future subclasses.

You are an expert coding assistant specialized in software refactoring, with many years of experience analyzing code transformations.

You will be given the diffs of a commit:

```
- **Diffs:**  
diff --git  
src/main/java/io/lettuce/core/resource/EventLoopResourcesWr  
apper.java  
src/main/java/io/lettuce/core/resource/EventLoopResourcesWr  
apper.java  
index 2a0193e6..bd1727fb 100644  
---  
src/main/java/io/lettuce/core/resource/EventLoopResourcesWr  
apper.java
```

```
+++
src/main/java/io/lettuce/core/resource/EventLoopResourcesWr
apper.java
@@ -13,7 +13,7 @@ import
io.netty.util.concurrent.EventExecutorGroup;
*
* @author Mark Paluch
*/
-class EventLoopResourcesWrapper implements
EventLoopResources {
+class EventLoopResourcesWrapper implements
EventLoopResources, IEventLoopResourcesWrapper {

    private final EventLoopResources delegate;

diff --git
src/main/java/io/lettuce/core/resource/IEventLoopResourcesW
rapper.java
src/main/java/io/lettuce/core/resource/IEventLoopResourcesW
rapper.java
new file mode 100644
index 0000000..4806d518
--- /dev/null
+++
src/main/java/io/lettuce/core/resource/IEventLoopResourcesW
rapper.java
@@ -0,0 +1,7 @@
+package io.lettuce.core.resource;
+
+import io.netty.util.concurrent.EventExecutorGroup;
+
+public interface IEventLoopResourcesWrapper {
+    boolean matches(Class<? extends EventExecutorGroup>
type);
+}
```

Your task is to identify which refactoring type(s) have been applied in transforming the original program into the new version. Use only the following list of predefined refactorings:

- (Add Method Parameter) – Introduces a new parameter to an existing method.
- (Remove Method Parameter) – Eliminates an existing parameter from a method signature.
- (Rename Method) – Changes the name of a method while preserving its behavior.
- (Rename Class) – Changes the name of a class without altering its structure.
- (Rename Package) – Changes the name of a package declaration.
- (Rename Field) – Changes the name of a class or instance variable.
- (Extract Class) – Moves a group of related fields and methods from an existing class into a newly created class.
- (Extract Superclass) – Creates a new superclass to encapsulate shared attributes and behavior from two or more existing classes.
- (Inline Method) – Replaces a method call with the method's body.
- (Pull Up Method) – Pull a method from a child class (subclass) in the original code to its parent class (superclass) in the transformed version.
- (Push Down Method) – Push a method from a parent class (superclass) in the original code to one or more of its child classes (subclasses) in the transformed version.
- (Pull Up Field) – Pull a field from a child class (subclass) in the original code to its parent class (superclass) in the transformed version.
- (Push Down Field) – Push a field from a parent class (superclass) in the original code to one or more of its child classes (subclasses) in the transformed version.

(Inline Class) – Merges a class into another when it is too small or redundant.

(Extract Interface) – Creates a new interface from existing method(s) in a class.

(Move Method) – Relocates a method to a more appropriate class.

(Move Field) – Relocates a field to a more appropriate class.

(Replace Magic Number with Constant) – Replaces a literal number with a named constant.

(Encapsulate Field) – Change a field access from not private to private and create get and set methods.

**\*\*Instructions:\*\***

1. Begin your response with a bullet-point list of the refactoring type(s) applied.
2. Then, briefly justify each identified refactoring with reference to the specific code changes.
3. Only include refactorings from the list above.
4. Be concise but precise in your explanations.

Do not generate explanations unrelated to the given transformation.



- Extract Interface

#### Justifications:

- **Extract Interface:**

- A new interface, `IEventLoopResourcesWrapper`, was created (new file  
`src/main/java/io/lettuce/core/resource/IEventLoopResourcesWrapper.java`)
- This new interface declares the method `boolean matches(Class<? extends EventExecutorGroup> type)`.
- The existing class `EventLoopResourcesWrapper` was modified to implement this new `IEventLoopResourcesWrapper` interface (its declaration changed from `class`

```
EventLoopResourcesWrapper implements EventLoopResources to class  
EventLoopResourcesWrapper implements EventLoopResources,  
IEventLoopResourcesWrapper ).
```

- This refactoring suggests that the `EventLoopResourcesWrapper` class provides an implementation for the `matches` method, and this part of its contract has been formalized into a separate interface, allowing other classes to potentially implement the same contract or for `EventLoopResourcesWrapper` to be used polymorphically via this new interface type.

You are an expert coding assistant specialized in software refactoring, with many years of experience analyzing code transformations.

You will be given the diffs of a commit:

```
- **Diffs:**  
diff --git  
src/main/java/io/lettuce/core/resource/EqualJitterDelay.java  
src/main/java/io/lettuce/core/resource/EqualJitterDelay.java  
index 03f448f3..643b4bfb 100644  
---  
src/main/java/io/lettuce/core/resource/EqualJitterDelay.java  
+++  
src/main/java/io/lettuce/core/resource/EqualJitterDelay.java  
@@ -51,7 +51,16 @@ class EqualJitterDelay extends  
ExponentialDelay {  
  
    @Override  
    public Duration createDelay(long attempt) {  
        -    long value = randomBetween(0, base *  
calculatePowerOfTwo(attempt));  
        +    long result;  
        +  
        +    if (attempt <= 0) { // safeguard against underflow  
        +        result = 0L;  
    }  
}
```

```

+     } else if (attempt >= 63) { // safeguard against overflow
in the bitshift operation
+
+         result = Long.MAX_VALUE - 1;
+
+     } else {
+
+         result = 1L << (attempt - 1);
+
+     }
+
+     long value = randomBetween(0, base * result);
+
return
applyBounds(Duration.ofNanos(targetTimeUnit.toNanos(value
))));
}

```

diff --git

src/main/java/io/lettuce/core/resource/ExponentialDelay.java

src/main/java/io/lettuce/core/resource/ExponentialDelay.java

index 73226249..95b29358 100644

---

src/main/java/io/lettuce/core/resource/ExponentialDelay.java

+++

src/main/java/io/lettuce/core/resource/ExponentialDelay.java

@@ -65,7 +65,16 @@ class ExponentialDelay extends Delay {

if (attempt <= 0) { // safeguard against underflow

delay = 0;

} else if (powersOf == 2) {

- delay = calculatePowerOfTwo(attempt);

+ long result;

+

+ if (attempt <= 0) { // safeguard against underflow

+ result = 0L;

+ } else if (attempt >= 63) { // safeguard against

overflow in the bitshift operation

+ result = Long.MAX\_VALUE - 1;

+ } else {

+ result = 1L << (attempt - 1);

+ }

+ delay = result;

```
        } else {
            delay = calculateAlternatePower(attempt);
        }
@@ -91,15 +100,5 @@ class ExponentialDelay extends Delay {
    }

    // fastpath with bitwise operator
-    protected static long calculatePowerOfTwo(long attempt) {
-
-
-        if (attempt <= 0) { // safeguard against underflow
-            return 0L;
-        } else if (attempt >= 63) { // safeguard against overflow in
the bitshift operation
-            return Long.MAX_VALUE - 1;
-        } else {
-            return 1L << (attempt - 1);
-        }
-    }

}
diff --git
src/main/java/io/lettuce/core/resource/FullJitterDelay.java
src/main/java/io/lettuce/core/resource/FullJitterDelay.java
index fa440651..9fc4abe8 100644
--- src/main/java/io/lettuce/core/resource/FullJitterDelay.java
+++ src/main/java/io/lettuce/core/resource/FullJitterDelay.java
@@ -57,7 +57,16 @@ class FullJitterDelay extends
ExponentialDelay {
    public Duration createDelay(long attempt) {

        long upperTarget =
targetTimeUnit.convert(upper.toNanos(),
TimeUnit.NANOSECONDS);
-
        long upperBase = base *
calculatePowerOfTwo(attempt);
+
        long result;
```

```

+
+     if (attempt <= 0) { // safeguard against underflow
+         result = 0L;
+     } else if (attempt >= 63) { // safeguard against overflow
in the bitshift operation
+         result = Long.MAX_VALUE - 1;
+     } else {
+         result = 1L << (attempt - 1);
+     }
+     long upperBase = base * result;
long temp = Math.min(upperTarget, 0 > upperBase ?
upperTarget : upperBase);
long delay = temp / 2 + randomBetween(0, temp / 2);
return
applyBounds(Duration.ofNanos(targetTimeUnit.toNanos(delay
)));

```

Your task is to identify which refactoring type(s) have been applied in transforming the original program into the new version. Use only the following list of predefined refactorings:

(Add Method Parameter) – Introduces a new parameter to an existing method.

(Remove Method Parameter) – Eliminates an existing parameter from a method signature.

(Rename Method) – Changes the name of a method while preserving its behavior.

(Rename Class) – Changes the name of a class without altering its structure.

(Rename Package) – Changes the name of a package declaration.

(Rename Field) – Changes the name of a class or instance variable.

(Extract Class) – Moves a group of related fields and methods from an existing class into a newly created class.

(Extract Superclass) – Creates a new superclass to encapsulate shared attributes and behavior from two or more existing classes.

(Inline Method) – Replaces a method call with the method's body.

(Pull Up Method) – Pull a method from a child class (subclass) in the original code to its parent class (superclass) in the transformed version.

(Push Down Method) – Push a method from a parent class (superclass) in the original code to one or more of its child classes (subclasses) in the transformed version.

(Pull Up Field) – Pull a field from a child class (subclass) in the original code to its parent class (superclass) in the transformed version.

(Push Down Field) – Push a field from a parent class (superclass) in the original code to one or more of its child classes (subclasses) in the transformed version.

(Inline Class) – Merges a class into another when it is too small or redundant.

(Extract Interface) – Creates a new interface from existing method(s) in a class.

(Move Method) – Relocates a method to a more appropriate class.

(Move Field) – Relocates a field to a more appropriate class.

(Replace Magic Number with Constant) – Replaces a literal number with a named constant.

(Encapsulate Field) – Change a field access from not private to private and create get and set methods.

**\*\*Instructions:\*\***

1. Begin your response with a bullet-point list of the refactoring type(s) applied.
2. Then, briefly justify each identified refactoring with reference to the specific code changes.
3. Only include refactorings from the list above.
4. Be concise but precise in your explanations.

Do not generate explanations unrelated to the given transformation.



⋮

- Inline Method

#### Justifications:

- Inline Method:

- The `protected static` method `calculatePowerOfTwo(long attempt)` was removed from the `ExponentialDelay` class.
- The body of this `calculatePowerOfTwo` method was directly inserted into its call sites. This occurred in:
  - The `createDelay` method of the `ExponentialDelay` class itself.
  - The `createDelay` method of the `EqualJitterDelay` class (a subclass of `ExponentialDelay`).
  - The `createDelay` method of the `FullJitterDelay` class (also a subclass of `ExponentialDelay`).
- In all these locations, the call to `calculatePowerOfTwo(attempt)` was replaced with the equivalent logic:

Java



```
long result;
if (attempt <= 0) { // safeguard against underflow
    result = 0L;
} else if (attempt >= 63) { // safeguard against overflow in the bits:
    result = Long.MAX_VALUE - 1;
} else {
    result = 1L << (attempt - 1);
}
// ... and then 'result' is used in place of the method call's return value
```

- This refactoring eliminates the `calculatePowerOfTwo` method by embedding its code directly where it was used, which can sometimes simplify the codebase if the method is small or only used in a few places with specific performance or clarity benefits from inlining.

You are an expert coding assistant specialized in software refactoring, with many years of experience analyzing code transformations.

You will be given the diffs of a commit:

```
- **Diffs:**  
diff --git  
src/main/java/io/lettuce/core/protocol/BaseRedisCommandBu  
lder.java  
src/main/java/io/lettuce/core/protocol/BaseRedisCommandBu  
lder.java  
index 98fc293e..4cdc81bc 100644  
---  
src/main/java/io/lettuce/core/protocol/BaseRedisCommandBu  
lder.java  
+++  
src/main/java/io/lettuce/core/protocol/BaseRedisCommandBu  
lder.java  
@@ -8,9 +8,11 @@ import io.lettuce.core.codec.RedisCodec;  
import io.lettuce.core.internal.LettuceAssert;  
import io.lettuce.core.output.*;  
  
+import java.net.SocketAddress;  
import java.nio.ByteBuffer;  
  
import static  
io.lettuce.core.protocol.CommandKeyword.LIMIT;  
+import static  
io.lettuce.core.protocol.CommandType.SENTINEL;  
  
/**
```

```

 * Common utility methods shared by all implementations of
the Redis command builder.

@@ -222,4 +224,9 @@ public class
BaseRedisCommandBuilder<K, V> {

    LettuceAssert.notEmpty(ranges, "Ranges " +
MUST_NOT_BE_NULL);

}

+
+  public Command<K, V, SocketAddress>
getMasterAddrByKey(K key) {
+
+      CommandArgs<K, V> args = new CommandArgs<>()
(codec).add("get-master-addr-by-name").addKey(key);
+
+      return createCommand(SENTINEL, new
SocketAddressOutput<>(codec), args);
+
}

diff --git
src/main/java/io/lettuce/core/resource/EEpollProvider.java
src/main/java/io/lettuce/core/resource/EEpollProvider.java
index d980efb7..cfa76e6b 100644
--- src/main/java/io/lettuce/core/resource/EEpollProvider.java
+++ src/main/java/io/lettuce/core/resource/EEpollProvider.java
@@ -6,10 +6,12 @@ import
io.netty.channel.epoll.EpollChannelOption;
import java.time.Duration;

public class EEpollProvider {

+
/***
 * Apply TcpUserTimeout options.
*/
public static void applyTcpUserTimeout(Bootstrap
bootstrap, Duration timeout) {

bootstrap.option(EpollChannelOption.TCP_USER_TIMEOUT,

```

```
Math.toIntExact(timeout.toMillis())));
}

+
}

diff --git
src/main/java/io/lettuce/core/resource/IEventLoopResourcesW
rapper.java
src/main/java/io/lettuce/core/resource/IEventLoopResourcesW
rapper.java
index 4806d518..f9185729 100644
---

src/main/java/io/lettuce/core/resource/IEventLoopResourcesW
rapper.java
+++  

src/main/java/io/lettuce/core/resource/IEventLoopResourcesW
rapper.java
@@ -3,5 +3,7 @@ package io.lettuce.core.resource;
import io.netty.util.concurrent.EventExecutorGroup;

public interface IEventLoopResourcesWrapper {
+
    boolean matches(Class<? extends EventExecutorGroup>
type);
+
}

diff --git
src/main/java/io/lettuce/core/sentinel/SentinelCommandBuilde
r.java
src/main/java/io/lettuce/core/sentinel/SentinelCommandBuilde
r.java
index 568f59ef..f52a0058 100644
---

src/main/java/io/lettuce/core/sentinel/SentinelCommandBuilde
r.java
+++  

src/main/java/io/lettuce/core/sentinel/SentinelCommandBuilde
```

```
r.java

@@ -22,7 +22,6 @@ package io.lettuce.core.sentinel;

import static io.lettuce.core.protocol.CommandKeyword.*;
import static io.lettuce.core.protocol.CommandType.*;

-import java.net.SocketAddress;
import java.util.List;
import java.util.Map;

@@ -34,7 +33,6 @@ import
io.lettuce.core.output.IntegerOutput;
import io.lettuce.core.output.KeyOutput;
import io.lettuce.core.output.ListOfMapsOutput;
import io.lettuce.core.output.MapOutput;
-import io.lettuce.core.output.SocketAddressOutput;
import io.lettuce.core.output.StatusOutput;
import io.lettuce.core.protocol.BaseRedisCommandBuilder;
import io.lettuce.core.protocol.Command;

@@ -52,11 +50,6 @@ class SentinelCommandBuilder<K, V>
extends BaseRedisCommandBuilder<K, V> {

    super(codec);
}

-
    public Command<K, V, SocketAddress>
getMasterAddrByKey(K key) {
    -
        CommandArgs<K, V> args = new CommandArgs<>
(codec).add("get-master-addr-by-name").addKey(key);
    -
        return createCommand(SENTINEL, new
SocketAddressOutput<>(codec), args);
    -
}

-
    public Command<K, V, List<Map<K, V>>> masters() {
        -
            CommandArgs<K, V> args = new CommandArgs<>
(codec).add("masters");
        -
            return createCommand(SENTINEL, new
ListOfMapsOutput<>(codec), args);
}
```

Your task is to identify which refactoring type(s) have been applied in transforming the original program into the new version. Use only the following list of predefined refactorings:

(Add Method Parameter) – Introduces a new parameter to an existing method.

(Remove Method Parameter) – Eliminates an existing parameter from a method signature.

(Rename Method) – Changes the name of a method while preserving its behavior.

(Rename Class) – Changes the name of a class without altering its structure.

(Rename Package) – Changes the name of a package declaration.

(Rename Field) – Changes the name of a class or instance variable.

(Extract Class) – Moves a group of related fields and methods from an existing class into a newly created class.

(Extract Superclass) – Creates a new superclass to encapsulate shared attributes and behavior from two or more existing classes.

(Inline Method) – Replaces a method call with the method's body.

(Pull Up Method) – Pull a method from a child class (subclass) in the original code to its parent class (superclass) in the transformed version.

(Push Down Method) – Push a method from a parent class (superclass) in the original code to one or more of its child classes (subclasses) in the transformed version.

(Pull Up Field) – Pull a field from a child class (subclass) in the original code to its parent class (superclass) in the transformed version.

(Push Down Field) – Push a field from a parent class (superclass) in the original code to one or more of its child

classes (subclasses) in the transformed version.

(Inline Class) – Merges a class into another when it is too small or redundant.

(Extract Interface) – Creates a new interface from existing method(s) in a class.

(Move Method) – Relocates a method to a more appropriate class.

(Move Field) – Relocates a field to a more appropriate class.

(Replace Magic Number with Constant) – Replaces a literal number with a named constant.

(Encapsulate Field) – Change a field access from not private to private and create get and set methods.

**\*\*Instructions:\*\***

1. Begin your response with a bullet-point list of the refactoring type(s) applied.
2. Then, briefly justify each identified refactoring with reference to the specific code changes.
3. Only include refactorings from the list above.
4. Be concise but precise in your explanations.

Do not generate explanations unrelated to the given transformation.



⋮

- Pull Up Method

#### Justifications:

- **Pull Up Method:**

- The method `public Command<K, V, SocketAddress> getMasterAddrByKey(K key)` was moved from the subclass `SentinelCommandBuilder` to its superclass `BaseRedisCommandBuilder`.
- In `SentinelCommandBuilder.java` (the subclass), this method was removed.
- An identical method (same signature, body, and visibility) was added to `BaseRedisCommandBuilder.java` (the superclass).

- This refactoring is used to move common behavior from a subclass to its superclass, reducing duplication and making the method available to other potential subclasses of `BaseRedisCommandBuilder`.
- Associated imports were also adjusted: `java.net.SocketAddress` was added to `BaseRedisCommandBuilder.java` and removed from `SentinelCommandBuilder.java` (along with `SocketAddressOutput`).

The changes in `EEventLoopProvider.java` and `IEventLoopResourcesWrapper.java` consist only of adding blank lines, which are formatting changes and not code refactorings from the provided list.

You are an expert coding assistant specialized in software refactoring, with many years of experience analyzing code transformations.

You will be given the diffs of a commit:

```
- **Diffs:**  
diff --git  
src/main/java/io/lettuce/core/sentinel/StatefulRedisSentinelCo  
nnectionImpl.java  
src/main/java/io/lettuce/core/sentinel/StatefulRedisSentinelCo  
nnectionImpl.java  
index 2eb20904..a2dfa0ed 100644  
---  
src/main/java/io/lettuce/core/sentinel/StatefulRedisSentinelCo  
nnectionImpl.java  
+++  
src/main/java/io/lettuce/core/sentinel/StatefulRedisSentinelCo  
nnectionImpl.java  
@@ -44,7 +44,7 @@ import static  
io.lettuce.core.ClientOptions.DEFAULT_JSON_PARSER;  
public class StatefulRedisSentinelConnectionImpl<K, V>  
extends RedisChannelHandler<K, V>  
implements StatefulRedisSentinelConnection<K, V> {
```

```

- protected final RedisCodec<K, V> codec;
+ protected final RedisCodec<K, V> redisCodec;

protected final RedisSentinelCommands<K, V> sync;

@@ -78,7 +78,7 @@ public class
StatefulRedisSentinelConnectionImpl<K, V> extends
RedisChannelHandl

super(writer, timeout);

-
this.codec = codec;
+
this.redisCodec = codec;
this.async = new RedisSentinelAsyncCommandsImpl<>(
this, codec);
this.sync = syncHandler(async,
RedisSentinelCommands.class);
this.reactive = new
RedisSentinelReactiveCommandsImpl<>(this, codec, parser);

```

Your task is to identify which refactoring type(s) have been applied in transforming the original program into the new version. Use only the following list of predefined refactorings:

- (Add Method Parameter) – Introduces a new parameter to an existing method.
- (Remove Method Parameter) – Eliminates an existing parameter from a method signature.
- (Rename Method) – Changes the name of a method while preserving its behavior.
- (Rename Class) – Changes the name of a class without altering its structure.
- (Rename Package) – Changes the name of a package declaration.

(Rename Field) – Changes the name of a class or instance variable.

(Extract Class) – Moves a group of related fields and methods from an existing class into a newly created class.

(Extract Superclass) – Creates a new superclass to encapsulate shared attributes and behavior from two or more existing classes.

(Inline Method) – Replaces a method call with the method's body.

(Pull Up Method) – Pull a method from a child class (subclass) in the original code to its parent class (superclass) in the transformed version.

(Push Down Method) – Push a method from a parent class (superclass) in the original code to one or more of its child classes (subclasses) in the transformed version.

(Pull Up Field) – Pull a field from a child class (subclass) in the original code to its parent class (superclass) in the transformed version.

(Push Down Field) – Push a field from a parent class (superclass) in the original code to one or more of its child classes (subclasses) in the transformed version.

(Inline Class) – Merges a class into another when it is too small or redundant.

(Extract Interface) – Creates a new interface from existing method(s) in a class.

(Move Method) – Relocates a method to a more appropriate class.

(Move Field) – Relocates a field to a more appropriate class.

(Replace Magic Number with Constant) – Replaces a literal number with a named constant.

(Encapsulate Field) – Change a field access from not private to private and create get and set methods.

**\*\*Instructions:\*\***

1. Begin your response with a bullet-point list of the refactoring type(s) applied.

2. Then, briefly justify each identified refactoring with reference to the specific code changes.
3. Only include refactorings from the list above.
4. Be concise but precise in your explanations.

Do not generate explanations unrelated to the given transformation.



⋮

- Rename Field

#### Justifications:

- Rename Field:

- The instance field `codec` within the `StatefulRedisSentinelConnectionImpl` class was renamed to `redisCodec`.
- The declaration changed from `protected final RedisCodec<K, V> codec;` to `protected final RedisCodec<K, V> redisCodec;`.
- The assignment in the constructor was also updated accordingly, from `this.codec = codec;` to `this.redisCodec = codec;` (where the right-hand side `codec` is a constructor parameter).

You are an expert coding assistant specialized in software refactoring, with many years of experience analyzing code transformations.

You will be given the diffs of a commit:

- **Diff:**

```
diff --git  
src/main/java/io/lettuce/core/dynamic/intercept/DefaultMetho  
dInvoker.java  
src/main/java/io/lettuce/core/dynamic/intercept/DefaultMetho  
dInvoker.java
```

```
index 624e047cc..1d0942a19 100644
---
src/main/java/io/lettuce/core/dynamic/intercept/DefaultMethodInvoker.java
+++  
src/main/java/io/lettuce/core/dynamic/intercept/DefaultMethodInvoker.java  
@@ -5,7 +5,6 @@ import java.lang.reflect.Method;  
import java.util.Map;  
import java.util.concurrent.ConcurrentHashMap;  
  
-import io.lettuce.core.internal.DefaultMethods;  
import io.lettuce.core.internal.LettuceAssert;  
  
/**  
@@ -35,16 +34,8 @@ public class DefaultMethodInvoker  
implements MethodInterceptor {  
  
    InvocationTargetException targetProvider =  
    (InvocationTargetException) invocation;  
  
    -    return methodHandleCache.computeIfAbsent(method,  
DefaultMethodInvoker::lookupMethodHandle)  
    +    return methodHandleCache.computeIfAbsent(method,  
MethodInterceptor::lookupMethodHandle)  
  
.bindTo(targetProvider.getInvocationTarget()).invokeWithArguments(invocation getArguments());  
}  
  
-    private static MethodHandle lookupMethodHandle(Method  
method) {  
-        try {  
-            return  
DefaultMethods.lookupMethodHandle(method);  
-        } catch (ReflectiveOperationException e) {
```

```
-        throw new IllegalArgumentException(e);
-
-    }
-
-}
-
}

diff --git
src/main/java/io/lettuce/core/dynamic/intercept/MethodInterce
ptor.java
src/main/java/io/lettuce/core/dynamic/intercept/MethodInterce
ptor.java
index 3df6a17ea..d372e0067 100644
---
src/main/java/io/lettuce/core/dynamic/intercept/MethodInterce
ptor.java
+++
src/main/java/io/lettuce/core/dynamic/intercept/MethodInterce
ptor.java
@@ -1,5 +1,10 @@
package io.lettuce.core.dynamic.intercept;

+import io.lettuce.core.internal.DefaultMethods;
+
+import java.lang.invoke.MethodHandle;
+import java.lang.reflect.Method;
+
+/*
 * Intercepts calls on an interface on its way to the target.
 * These are nested "on top" of the target.
 *
@@ -11,6 +16,14 @@ package
io.lettuce.core.dynamic.intercept;
*/
public interface MethodInterceptor {

+    static MethodHandle lookupMethodHandle(Method
method) {
```

```

+     try {
+         return
DefaultMethods.lookupMethodHandle(method);
+     } catch (ReflectiveOperationException e) {
+         throw new IllegalArgumentException(e);
+     }
+
+ /**
* Implement this method to perform extra treatments
before and after the invocation. Polite implementations would
certainly
* like to invoke {@link MethodInvocation#proceed()}.

```

Your task is to identify which refactoring type(s) have been applied in transforming the original program into the new version. Use only the following list of predefined refactorings:

(Add Method Parameter) – Introduces a new parameter to an existing method.

(Remove Method Parameter) – Eliminates an existing parameter from a method signature.

(Rename Method) – Changes the name of a method while preserving its behavior.

(Rename Class) – Changes the name of a class without altering its structure.

(Rename Package) – Changes the name of a package declaration.

(Rename Field) – Changes the name of a class or instance variable.

(Extract Class) – Moves a group of related fields and methods from an existing class into a newly created class.

(Extract Superclass) – Creates a new superclass to encapsulate shared attributes and behavior from two or more existing classes.

(Inline Method) – Replaces a method call with the method's body.

(Pull Up Method) – Pull a method from a child class (subclass) in the original code to its parent class (superclass) in the transformed version.

(Push Down Method) – Push a method from a parent class (superclass) in the original code to one or more of its child classes (subclasses) in the transformed version.

(Pull Up Field) – Pull a field from a child class (subclass) in the original code to its parent class (superclass) in the transformed version.

(Push Down Field) – Push a field from a parent class (superclass) in the original code to one or more of its child classes (subclasses) in the transformed version.

(Inline Class) – Merges a class into another when it is too small or redundant.

(Extract Interface) – Creates a new interface from existing method(s) in a class.

(Move Method) – Relocates a method to a more appropriate class.

(Move Field) – Relocates a field to a more appropriate class.

(Replace Magic Number with Constant) – Replaces a literal number with a named constant.

(Encapsulate Field) – Change a field access from not private to private and create get and set methods.

**\*\*Instructions:\*\***

1. Begin your response with a bullet-point list of the refactoring type(s) applied.
2. Then, briefly justify each identified refactoring with reference to the specific code changes.
3. Only include refactorings from the list above.
4. Be concise but precise in your explanations.

Do not generate explanations unrelated to the given transformation.



- Move Method

#### Justifications:

- Move Method:

- The `static` method `lookupMethodHandle(Method method)` was moved from the class `DefaultMethodInvoker` to the interface `MethodInterceptor`.
- In `DefaultMethodInvoker.java`, the `private static MethodHandle lookupMethodHandle(Method method)` was removed.
- An identical method (same logic and parameters, becoming an implicitly public static method) was added to the `MethodInterceptor.java` interface.
- The method reference in `DefaultMethodInvoker.java` that used this method was updated from `DefaultMethodInvoker::lookupMethodHandle` to `MethodInterceptor::lookupMethodHandle`, reflecting the method's new location.
- This refactoring relocates a helper method to an interface with which it is logically associated, making it potentially accessible to other implementers or users of the interface.

You are an expert coding assistant specialized in software refactoring, with many years of experience analyzing code transformations.

You will be given the diffs of a commit:

```
- **Diffs:**  
diff --git  
src/main/java/io/lettuce/core/dynamic/output/OutputRegistry.j  
ava  
src/main/java/io/lettuce/core/dynamic/output/OutputRegistry.j  
ava  
index 8d59004cf..b0fb03123 100644
```

```
---  
src/main/java/io/lettuce/core/dynamic/output/OutputRegistry.j  
ava  
+++  
src/main/java/io/lettuce/core/dynamic/output/OutputRegistry.j  
ava  
@@ -21,7 +21,7 @@ import io.lettuce.core.output.*;  
 * @see CommandOutput  
 */  
@SuppressWarnings("rawtypes")  
-public class OutputRegistry {  
+public class OutputRegistry implements  
OutputRegistryAbstract {  
  
    private static final Map<OutputType,  
CommandOutputFactory> BUILTIN = new LinkedHashMap<>();  
  
@@ -86,8 +86,9 @@ public class OutputRegistry {  
    * @param commandOutputClass must not be {@code null}.  
    * @param commandOutputFactory must not be {@code null}.  
    */  
+    @Override  
    public <T extends CommandOutput<?, ?, ?>> void  
register(Class<T> commandOutputClass,  
-        CommandOutputFactory commandOutputFactory) {  
+        CommandOutputFactory  
commandOutputFactory) {  
  
    LettuceAssert.notNull(commandOutputClass,  
"CommandOutput class must not be null");  
    LettuceAssert.notNull(commandOutputFactory,  
"CommandOutputFactory must not be null");  
@@ -217,37 +218,4 @@ public class OutputRegistry {  
};
```

```
}

-    @SuppressWarnings("serial")
-    static class CodecVariableTypeResolver implements
ResolvableType.VariableResolver {

-
-
-    private final TypeInformation<?> codecType;
-
-
-    private final List<TypeInformation<?>> typeArguments;
-
-
-    public CodecVariableTypeResolver(TypeInformation<?>
codecType) {
-
-
-        this.codecType =
codecType.getSuperTypeInformation(RedisCodec.class);
-        this.typeArguments =
this.codecType.getTypeArguments();
-    }
-
-
-    @Override
-    public Object getSource() {
-        return codecType;
-    }
-
-
-    @Override
-    public ResolvableType resolveVariable(TypeVariable<?>
variable) {
-
-
-        if (variable.getName().equals("K")) {
-            return
ResolvableType.forClass(typeArguments.get(0).getType());
-        }
-
-
-        if (variable.getName().equals("V")) {
-            return
ResolvableType.forClass(typeArguments.get(1).getType());
-
```

```
-      }
-      return null;
-    }
-
-  }
-
}

diff --git
src/main/java/io/lettuce/core/dynamic/output/OutputRegistryA
bstract.java
src/main/java/io/lettuce/core/dynamic/output/OutputRegistryA
bstract.java
new file mode 100644
index 00000000..07f2b976c
--- /dev/null
+++ 
src/main/java/io/lettuce/core/dynamic/output/OutputRegistryA
bstract.java
@@ -0,0 +1,47 @@
+package io.lettuce.core.dynamic.output;
+
+import io.lettuce.core.codec.RedisCodec;
+import io.lettuce.core.dynamic.support.ResolvableType;
+import io.lettuce.core.dynamic.support.TypeInformation;
+import io.lettuce.core.output.CommandOutput;
+
+import java.lang.reflect.TypeVariable;
+import java.util.List;
+
+
+public interface OutputRegistryAbstract {
+  <T extends CommandOutput<?, ?, ?>> void
register(Class<T> commandOutputClass,
+
+          CommandOutputFactory
commandOutputFactory);
+
+
+  @SuppressWarnings("serial")
```

```
+ public static class CodecVariableTypeResolver implements
ResolvableType.VariableResolver {
+
+
+     private final TypeInformation<?> codecType;
+
+
+     private final List<TypeInformation<?>> typeArguments;
+
+
+     public CodecVariableTypeResolver(TypeInformation<?>
codecType) {
+
+         this.codecType =
codecType.getSuperTypeInformation(RedisCodec.class);
+         this.typeArguments =
this.codecType.getTypeArguments();
+
+     }
+
+
+     @Override
+     public Object getSource() {
+         return codecType;
+     }
+
+
+     @Override
+     public ResolvableType resolveVariable(TypeVariable<?>
variable) {
+
+         if (variable.getName().equals("K")) {
+             return
ResolvableType.forName(typeArguments.get(0).getType());
+
+         }
+
+
+         if (variable.getName().equals("V")) {
+             return
ResolvableType.forName(typeArguments.get(1).getType());
+
+         }
+
+         return null;
+
+     }
}
```

```
+  
+  }  
+}
```

Your task is to identify which refactoring type(s) have been applied in transforming the original program into the new version. Use only the following list of predefined refactorings:

- (Add Method Parameter) – Introduces a new parameter to an existing method.
- (Remove Method Parameter) – Eliminates an existing parameter from a method signature.
- (Rename Method) – Changes the name of a method while preserving its behavior.
- (Rename Class) – Changes the name of a class without altering its structure.
- (Rename Package) – Changes the name of a package declaration.
- (Rename Field) – Changes the name of a class or instance variable.
- (Extract Class) – Moves a group of related fields and methods from an existing class into a newly created class.
- (Extract Superclass) – Creates a new superclass to encapsulate shared attributes and behavior from two or more existing classes.
- (Inline Method) – Replaces a method call with the method's body.
- (Pull Up Method) – Pull a method from a child class (subclass) in the original code to its parent class (superclass) in the transformed version.
- (Push Down Method) – Push a method from a parent class (superclass) in the original code to one or more of its child classes (subclasses) in the transformed version.
- (Pull Up Field) – Pull a field from a child class (subclass) in the original code to its parent class (superclass) in the

transformed version.

(Push Down Field) – Push a field from a parent class (superclass) in the original code to one or more of its child classes (subclasses) in the transformed version.

(Inline Class) – Merges a class into another when it is too small or redundant.

(Extract Interface) – Creates a new interface from existing method(s) in a class.

(Move Method) – Relocates a method to a more appropriate class.

(Move Field) – Relocates a field to a more appropriate class.

(Replace Magic Number with Constant) – Replaces a literal number with a named constant.

(Encapsulate Field) – Change a field access from not private to private and create get and set methods.

**\*\*Instructions:\*\***

1. Begin your response with a bullet-point list of the refactoring type(s) applied.
2. Then, briefly justify each identified refactoring with reference to the specific code changes.
3. Only include refactorings from the list above.
4. Be concise but precise in your explanations.

Do not generate explanations unrelated to the given transformation.



⋮

- Extract Interface

#### Justifications:

- **Extract Interface:**

- A new interface, `OutputRegistryAbstract`, was created (new file

```
src/main/java/io/lettuce/core/dynamic/output/OutputRegistryAbstract.java  
a).
```

- This new interface declares the method signature `<T extends CommandOutput<?, ?, ?>> void register(Class<T> commandOutputClass, CommandOutputFactory commandOutputFactory)`, which matches an existing public method in the `OutputRegistry` class.
- The existing class `OutputRegistry` was modified to implement this new `OutputRegistryAbstract` interface (its declaration changed from `public class OutputRegistry` to `public class OutputRegistry implements OutputRegistryAbstract`). The `register` method in `OutputRegistry` is now (implicitly or explicitly) marked with `@Override`.
- Additionally, the `static class CodecVariableTypeResolver`, which was formerly an inner class of `OutputRegistry`, was removed from `OutputRegistry.java` and moved into the new `OutputRegistryAbstract.java` interface, becoming a static nested class of the interface. This indicates that the helper class is considered part of the extracted abstraction.
- This refactoring formalizes a part of `OutputRegistry`'s contract and its associated helper types into a separate interface, promoting abstraction.

You are an expert coding assistant specialized in software refactoring, with many years of experience analyzing code transformations.

You will be given the diffs of a commit:

```
- **Diffs:**  
diff --git  
src/main/java/io/lettuce/core/dynamic/output/IOutputSelector.j  
ava  
src/main/java/io/lettuce/core/dynamic/output/IOutputSelector.j  
ava  
new file mode 100644  
index 00000000..835650904  
--- /dev/null  
+++  
src/main/java/io/lettuce/core/dynamic/output/IOutputSelector.j  
ava
```

```
@@ -0,0 +1,10 @@
+package io.lettuce.core.dynamic.output;
+
+import io.lettuce.core.codec.RedisCodec;
+import io.lettuce.core.dynamic.support.ResolvableType;
+
+public interface IOutputSelector {
+    ResolvableType getOutputType();
+
+    RedisCodec<?, ?> getRedisCodec();
+}
diff --git
src/main/java/io/lettuce/core/dynamic/output/OutputSelector.j
ava
src/main/java/io/lettuce/core/dynamic/output/OutputSelector.j
ava
index baf034cdf..d3813bc2d 100644
---
src/main/java/io/lettuce/core/dynamic/output/OutputSelector.j
ava
+++  

src/main/java/io/lettuce/core/dynamic/output/OutputSelector.j
ava
@@ -13,7 +13,7 @@ import
io.lettuce.core.internal.LettuceAssert;
 * @author Mark Paluch
 * @since 5.0
 */
-public class OutputSelector {
+public class OutputSelector implements IOutputSelector {

    private final ResolvableType outputType;

@@ -37,6 +37,7 @@ public class OutputSelector {
    /**
     * @return the output type.

```

```
 */
+ @Override
    public ResolvableType getOutputType() {
        return outputType;
    }
@@ -45,6 +46,7 @@ public class OutputSelector {
 *
 * @return the associated codec.
 */
+ @Override
    public RedisCodec<?, ?> getRedisCodec() {
        return redisCodec;
    }
}
```

Your task is to identify which refactoring type(s) have been applied in transforming the original program into the new version. Use only the following list of predefined refactorings:

(Add Method Parameter) – Introduces a new parameter to an existing method.

(Remove Method Parameter) – Eliminates an existing parameter from a method signature.

(Rename Method) – Changes the name of a method while preserving its behavior.

(Rename Class) – Changes the name of a class without altering its structure.

(Rename Package) – Changes the name of a package declaration.

(Rename Field) – Changes the name of a class or instance variable.

(Extract Class) – Moves a group of related fields and methods from an existing class into a newly created class.

(Extract Superclass) – Creates a new superclass to encapsulate shared attributes and behavior from two or more existing classes.

(Inline Method) – Replaces a method call with the method's body.

(Pull Up Method) – Pull a method from a child class (subclass) in the original code to its parent class (superclass) in the transformed version.

(Push Down Method) – Push a method from a parent class (superclass) in the original code to one or more of its child classes (subclasses) in the transformed version.

(Pull Up Field) – Pull a field from a child class (subclass) in the original code to its parent class (superclass) in the transformed version.

(Push Down Field) – Push a field from a parent class (superclass) in the original code to one or more of its child classes (subclasses) in the transformed version.

(Inline Class) – Merges a class into another when it is too small or redundant.

(Extract Interface) – Creates a new interface from existing method(s) in a class.

(Move Method) – Relocates a method to a more appropriate class.

(Move Field) – Relocates a field to a more appropriate class.

(Replace Magic Number with Constant) – Replaces a literal number with a named constant.

(Encapsulate Field) – Change a field access from not private to private and create get and set methods.

**\*\*Instructions:\*\***

1. Begin your response with a bullet-point list of the refactoring type(s) applied.
2. Then, briefly justify each identified refactoring with reference to the specific code changes.
3. Only include refactorings from the list above.
4. Be concise but precise in your explanations.

Do not generate explanations unrelated to the given transformation.



- Extract Interface

#### Justifications:

- Extract Interface:

- A new interface, `IOutputSelector`, was created (new file `src/main/java/io/lettuce/core/dynamic/output/IOutputSelector.java`).
- This new interface declares the methods `ResolvableType getOutputType()` and `RedisCodec<?, ?> getRedisCodec()`. These signatures match existing public methods in the `OutputSelector` class.
- The existing class `OutputSelector` was modified to implement this new `IOutputSelector` interface (its declaration changed from `public class OutputSelector` to `public class OutputSelector implements IOutputSelector`).
- The corresponding methods `getOutputType()` and `getRedisCodec()` in the `OutputSelector` class are now annotated with `@Override`, indicating they are implementations of the methods defined in the new interface.
- This refactoring is used to define a common type for classes that can select output types and codecs, promoting abstraction and polymorphism.

You are an expert coding assistant specialized in software refactoring, with many years of experience analyzing code transformations.

You will be given the diffs of a commit:

```
- **Diffs:**  
diff --git  
src/main/java/io/lettuce/core/dynamic/segment/AnnotationCo  
mmandSegmentFactory.java  
src/main/java/io/lettuce/core/dynamic/segment/AnnotationCo  
mmandSegmentFactory.java
```

```
index 151aaccda..de1eb3f63 100644
---
src/main/java/io/lettuce/core/dynamic/segment/AnnotationCo
mmandSegmentFactory.java
+++
src/main/java/io/lettuce/core/dynamic/segment/AnnotationCo
mmandSegmentFactory.java
@@ -107,24 +107,6 @@ public class
AnnotationCommandSegmentFactory implements
CommandSegmentFactory {
    return parts;
}

- private LetterCase getLetterCase(CommandMethod
commandMethod) {
-
-
-     if
(commandMethod.hasAnnotation(CommandNaming.class)) {
-
-         LetterCase letterCase =
commandMethod.getMethod().getAnnotation(CommandNami
ng.class).letterCase();
-
-         if (letterCase != LetterCase.DEFAULT) {
-
-             return letterCase;
-
-         }
-
-     }
-
-
-     Class<?> declaringClass =
commandMethod.getMethod().getDeclaringClass();
-
-     CommandNaming annotation =
declaringClass.getAnnotation(CommandNaming.class);
-
-     if (annotation != null && annotation.letterCase() !=
LetterCase.DEFAULT) {
-
-         return annotation.letterCase();
-
-     }
-
-
-     return LetterCase.UPPERCASE;
}
```

```
-    }

-
     private Strategy getNamingStrategy(CommandMethod
commandMethod) {

    if
(commandMethod.hasAnnotation(CommandNaming.class)) {
diff --git
src/main/java/io/lettuce/core/dynamic/segment/CommandSeg
mentFactory.java
src/main/java/io/lettuce/core/dynamic/segment/CommandSeg
mentFactory.java
index 308651405..3db16311d 100644
---
src/main/java/io/lettuce/core/dynamic/segment/CommandSeg
mentFactory.java
+++  

src/main/java/io/lettuce/core/dynamic/segment/CommandSeg
mentFactory.java
@@ -1,6 +1,7 @@
 package io.lettuce.core.dynamic.segment;

import io.lettuce.core.dynamic.CommandMethod;
+import io.lettuce.core.dynamic.annotation.CommandNaming;

/**
 * Strategy interface to create {@link CommandSegments} for
a {@link CommandMethod}.
@@ -18,4 +19,21 @@ public interface
CommandSegmentFactory {

    */

    CommandSegments
createCommandSegments(CommandMethod
commandMethod);
```

```

+ default CommandNaming.LetterCase
getLetterCase(CommandMethod commandMethod) {
+
+
+ if
(commandMethod.hasAnnotation(CommandNaming.class)) {
+
+     CommandNaming.LetterCase letterCase =
commandMethod.getMethod().getAnnotation(CommandNami
ng.class).letterCase();
+
+     if (letterCase !=

CommandNaming.LetterCase.DEFAULT) {
+
+         return letterCase;
+
+     }
+
+ }
+
+
+ Class<?> declaringClass =
commandMethod.getMethod().getDeclaringClass();
+
+ CommandNaming annotation =
declaringClass.getAnnotation(CommandNaming.class);
+
+ if (annotation != null && annotation.letterCase() !=

CommandNaming.LetterCase.DEFAULT) {
+
+     return annotation.letterCase();
+
+ }
+
+
+ return CommandNaming.LetterCase.UPPERCASE;
+
+ }
}

```

Your task is to identify which refactoring type(s) have been applied in transforming the original program into the new version. Use only the following list of predefined refactorings:

(Add Method Parameter) – Introduces a new parameter to an existing method.

(Remove Method Parameter) – Eliminates an existing parameter from a method signature.

(Rename Method) – Changes the name of a method while preserving its behavior.

(Rename Class) – Changes the name of a class without altering its structure.

(Rename Package) – Changes the name of a package declaration.

(Rename Field) – Changes the name of a class or instance variable.

(Extract Class) – Moves a group of related fields and methods from an existing class into a newly created class.

(Extract Superclass) – Creates a new superclass to encapsulate shared attributes and behavior from two or more existing classes.

(Inline Method) – Replaces a method call with the method's body.

(Pull Up Method) – Pull a method from a child class (subclass) in the original code to its parent class (superclass) in the transformed version.

(Push Down Method) – Push a method from a parent class (superclass) in the original code to one or more of its child classes (subclasses) in the transformed version.

(Pull Up Field) – Pull a field from a child class (subclass) in the original code to its parent class (superclass) in the transformed version.

(Push Down Field) – Push a field from a parent class (superclass) in the original code to one or more of its child classes (subclasses) in the transformed version.

(Inline Class) – Merges a class into another when it is too small or redundant.

(Extract Interface) – Creates a new interface from existing method(s) in a class.

(Move Method) – Relocates a method to a more appropriate class.

(Move Field) – Relocates a field to a more appropriate class.

(Replace Magic Number with Constant) – Replaces a literal number with a named constant.

(Encapsulate Field) – Change a field access from not private to private and create get and set methods.

**\*\*Instructions:\*\***

1. Begin your response with a bullet-point list of the refactoring type(s) applied.
2. Then, briefly justify each identified refactoring with reference to the specific code changes.
3. Only include refactorings from the list above.
4. Be concise but precise in your explanations.

Do not generate explanations unrelated to the given transformation.



⋮

- Pull Up Method

#### Justifications:

- **Pull Up Method:**

- The method `getLetterCase(CommandMethod commandMethod)` was moved from the class `AnnotationCommandSegmentFactory` to the `CommandSegmentFactory` interface, which `AnnotationCommandSegmentFactory` implements.
- In `AnnotationCommandSegmentFactory.java`, the `private LetterCase getLetterCase(CommandMethod commandMethod)` method (where `LetterCase` is effectively `CommandNaming.LetterCase`) was removed.
- An identical method in terms of logic and parameters, `default CommandNaming.LetterCase getLetterCase(CommandMethod commandMethod)`, was added to the `CommandSegmentFactory.java` interface as a default method.
- This refactoring moves a method from an implementing class to its interface as a default method, allowing the behavior to be shared among all implementers of the interface or providing a common default implementation. The method became implicitly public as part of the interface.

You are an expert coding assistant specialized in software refactoring, with many years of experience analyzing code transformations.

You will be given the diffs of a commit:

- \*\*Diffs:\*\*

diff --git

src/main/java/io/lettuce/core/internal/ExceptionFactory.java  
src/main/java/io/lettuce/core/internal/ExceptionFactory.java  
index b4374082a..69c15bc17 100644

---

src/main/java/io/lettuce/core/internal/ExceptionFactory.java

+++

src/main/java/io/lettuce/core/internal/ExceptionFactory.java  
@@ -86,7 +86,7 @@ public abstract class ExceptionFactory {  
 return MINUTES.format(time);

}

- if (isExactSeconds(duration)) {

+ if (duration.toMillis() % (1000) == 0 &&

duration.getNano() == 0) {

return SECONDS.format(time);

}

@@ -101,10 +101,6 @@ public abstract class ExceptionFactory

{

return duration.toMillis() % (1000 \* 60) == 0 &&

duration.getNano() == 0;

}

- private static boolean isExactSeconds(Duration duration) {

- return duration.toMillis() % (1000) == 0 &&

duration.getNano() == 0;

- }

```
- private static boolean isExactMillis(Duration duration) {  
    return duration.toNanos() % (1000 * 1000) == 0;  
}
```

Your task is to identify which refactoring type(s) have been applied in transforming the original program into the new version. Use only the following list of predefined refactorings:

(Add Method Parameter) – Introduces a new parameter to an existing method.

(Remove Method Parameter) – Eliminates an existing parameter from a method signature.

(Rename Method) – Changes the name of a method while preserving its behavior.

(Rename Class) – Changes the name of a class without altering its structure.

(Rename Package) – Changes the name of a package declaration.

(Rename Field) – Changes the name of a class or instance variable.

(Extract Class) – Moves a group of related fields and methods from an existing class into a newly created class.

(Extract Superclass) – Creates a new superclass to encapsulate shared attributes and behavior from two or more existing classes.

(Inline Method) – Replaces a method call with the method's body.

(Pull Up Method) – Pull a method from a child class (subclass) in the original code to its parent class (superclass) in the transformed version.

(Push Down Method) – Push a method from a parent class (superclass) in the original code to one or more of its child classes (subclasses) in the transformed version.

(Pull Up Field) – Pull a field from a child class (subclass) in the original code to its parent class (superclass) in the transformed version.

(Push Down Field) – Push a field from a parent class (superclass) in the original code to one or more of its child classes (subclasses) in the transformed version.

(Inline Class) – Merges a class into another when it is too small or redundant.

(Extract Interface) – Creates a new interface from existing method(s) in a class.

(Move Method) – Relocates a method to a more appropriate class.

(Move Field) – Relocates a field to a more appropriate class.

(Replace Magic Number with Constant) – Replaces a literal number with a named constant.

(Encapsulate Field) – Change a field access from not private to private and create get and set methods.

**\*\*Instructions:\*\***

1. Begin your response with a bullet-point list of the refactoring type(s) applied.
2. Then, briefly justify each identified refactoring with reference to the specific code changes.
3. Only include refactorings from the list above.
4. Be concise but precise in your explanations.

Do not generate explanations unrelated to the given transformation.



⋮

- Inline Method

#### Justifications:

- **Inline Method:**

- The `private static boolean isExactSeconds(Duration duration)` method was removed from the `ExceptionFactory` class.

- The body of the `isExactSeconds` method, which was `duration.toMillis() % (1000) == 0 && duration.getNano() == 0`, was directly substituted for the call to `isExactSeconds(duration)` within the `formatDuration` method.
- Specifically, the condition `if (isExactSeconds(duration))` in `formatDuration` was changed to `if (duration.toMillis() % (1000) == 0 && duration.getNano() == 0)`.
- This refactoring replaces a method call with the method's actual code, which is often done when a method's body is simple or to eliminate an unnecessary level of indirection.

You are an expert coding assistant specialized in software refactoring, with many years of experience analyzing code transformations.

You will be given the diffs of a commit:

```
- **Diffs:**  
diff --git  
src/main/java/io/lettuce/core/dynamic/output/OutputRegistry.j  
ava  
src/main/java/io/lettuce/core/dynamic/output/OutputRegistry.j  
ava  
index b0fb03123..bafe7eaa1 100644  
---  
src/main/java/io/lettuce/core/dynamic/output/OutputRegistry.j  
ava  
+++  
src/main/java/io/lettuce/core/dynamic/output/OutputRegistry.j  
ava  
@@ -1,6 +1,5 @@  
package io.lettuce.core.dynamic.output;  
  
-import java.lang.reflect.TypeVariable;  
import java.util.ArrayList;
```

```
import java.util.LinkedHashMap;
import java.util.List;
@@ -23,8 +22,6 @@ import io.lettuce.core.output.*;
@SuppressWarnings("rawtypes")
public class OutputRegistry implements
OutputRegistryAbstract {

    - private static final Map<OutputType,
CommandOutputFactory> BUILTIN = new LinkedHashMap<>();

    -
        private final Map<OutputType, CommandOutputFactory>
registry = new LinkedHashMap<>();

    static {
@@ -58,7 +55,7 @@ public class OutputRegistry implements
OutputRegistryAbstract {

        register(registry, StringMatchResultOutput.class,
StringMatchResultOutput::new);

    -
        BUILTIN.putAll(registry);
    +
        OutputRegistryAbstract.BUILTIN.putAll(registry);
    }

    /**
@@ -76,7 +73,7 @@ public class OutputRegistry implements
OutputRegistryAbstract {
    public OutputRegistry(boolean registerBuiltin) {

        if (registerBuiltin) {
    -
        registry.putAll(BUILTIN);
    +
        registry.putAll(OutputRegistryAbstract.BUILTIN);
    }

    }
```

```
diff --git
src/main/java/io/lettuce/core/dynamic/output/OutputRegistryA
bstract.java
src/main/java/io/lettuce/core/dynamic/output/OutputRegistryA
bstract.java
index 07f2b976c..a16182f07 100644
---
src/main/java/io/lettuce/core/dynamic/output/OutputRegistryA
bstract.java
+++  

src/main/java/io/lettuce/core/dynamic/output/OutputRegistryA
bstract.java
@@ -6,9 +6,13 @@ import
io.lettuce.core.dynamic.support.TypeInformation;
import io.lettuce.core.output.CommandOutput;

import java.lang.reflect.TypeVariable;
+import java.util.LinkedHashMap;
import java.util.List;
+import java.util.Map;

public interface OutputRegistryAbstract {
+  Map<OutputType, CommandOutputFactory> BUILTIN =
new LinkedHashMap<>();
+
  <T extends CommandOutput<?, ?, ?>> void
register(Class<T> commandOutputClass,
          CommandOutputFactory
commandOutputFactory);
```

Your task is to identify which refactoring type(s) have been applied in transforming the original program into the new version. Use only the following list of predefined refactorings:

(Add Method Parameter) – Introduces a new parameter to an existing method.

(Remove Method Parameter) – Eliminates an existing parameter from a method signature.

(Rename Method) – Changes the name of a method while preserving its behavior.

(Rename Class) – Changes the name of a class without altering its structure.

(Rename Package) – Changes the name of a package declaration.

(Rename Field) – Changes the name of a class or instance variable.

(Extract Class) – Moves a group of related fields and methods from an existing class into a newly created class.

(Extract Superclass) – Creates a new superclass to encapsulate shared attributes and behavior from two or more existing classes.

(Inline Method) – Replaces a method call with the method's body.

(Pull Up Method) – Pull a method from a child class (subclass) in the original code to its parent class (superclass) in the transformed version.