Running Examples

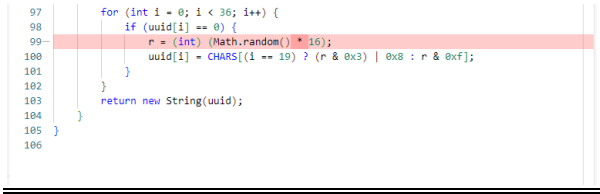
This document shows 5 running examples that we encountered during the experiment. Each of the examples are listed with

- artifact: the name of the target java source code project
- tpl_name: the name of the detected TPL
- cur_class: the path of file that copies
- lib_class: the path of file being copied
- complexity: the sum of function complexities
- pagerank: pagerank of the TPL class
- percentile: percentile of the TPL class. 0 is the top, 100 is the bottom.
- av: TPL artifact name and verion
- Comments: our comments of the example

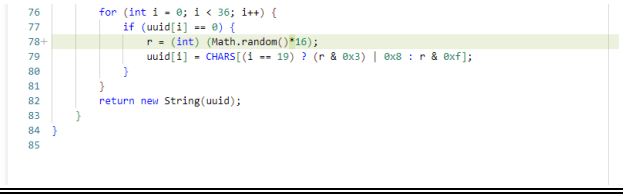
Example 1

artifact: geosdi@geo-platform,
tpl_name: org.uberfire:uberfire-wires-core-api,
cur_class: UUIDGenerator@/home/xxxx/projects/github_mvn_1k/geosdi@geo-platform/geoplatform-gui/core/geoplatform-api/src/main/java/org/geosdi/geoplatform/gui/model/UUIDGenerator.java,
lib_class: UUID@/home/xxxx/projects/mvn_critical_lib/org.uberfire/uberfire-wires-core-api/0.9.0.Final/org/uberfire/ext/wires/core/api/shapes/UUID.java,
complexity: 149.14806106545882,
pagerank: 0.015483587800840762,
percentile: 0.08823529411764706,
class_id: UUID@/home/xxxx/projects/mvn_critical_lib/org.uberfire/uberfire-wires-core-api/0.9.0.Final/org/uberfire/ext/wires/core/api/shapes/UUID.java,
av: uberfire-wires-core-api:0.9.0.Final,
Comments:
The method content are mostly the same, but the original author information was removed.


UUID.java
/*
*
* geo-platform
* Rich webgis framework
* http://geo-platform.org
* ==
*
* Copyright (C) 2008-2024 geoSDI Group (CNR IMAA - Potenza - ITALY).
*
* This program is free software: you can redistribute it and/or modify it
* under the terms of the GNU General Public License as published by
* the Free Software Foundation, either version 3 of the License, or
* (at your option) any later version. This program is distributed in the
* hope that it will be useful, but WITHOUT ANY WARRANTY; without
* even the implied warranty of MERCHANTABILITY or FITNESS FOR
* A PARTICULAR PURPOSE. See the GNU General Public License
* for more details. You should have received a copy of the GNU General
* Public License along with this program. If not, see http://www.gnu.org/licenses/
*
* ==
*
* Linking this library statically or dynamically with other modules is
* making a combined work based on this library. Thus, the terms and
* conditions of the GNU General Public License cover the whole combination.
*
* As a special exception, the copyright holders of this library give you permission
* to link this library with independent modules to produce an executable, regardless
* of the license terms of these independent modules, and to copy and distribute
* the resulting executable under terms of your choice, provided that you also meet,
* for each linked independent module, the terms and conditions of the license of
* that module. An independent module is a module which is not derived from or
* based on this library. If you modify this library, you may extend this exception
* to your version of the library, but you are not obligated to do so. If you do not
* wish to do so, delete this exception statement from your version.
*/
package org.geosdi.geoplatform.gui.model;
/**
* @author Nazzareno Sileno - CNR IMAA geoSDI Group
* @email nazzareno.sileno@geosdi.org
*/
public class UUIDGenerator {
private static final char[] CHARS = "0123456789ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmn
/**
* Generate a random uuid of the specified length. Example: uuid(15) returns
* "VcydxgltxrVZSTV"
*
* @param len
* the desired number of characters
*/
public static String uuid(int len) {
return uuid(len, CHARS.length);
/*
* Copyright 2015 JBoss, by Red Hat, Inc
*
* Licensed under the Apache License, Version 2.0 (the "License");
* you may not use this file except in compliance with the License.
* You may obtain a copy of the License at
*
*   http://www.apache.org/licenses/LICENSE-2.0
*
* Unless required by applicable law or agreed to in writing, software
* distributed under the License is distributed on an "AS IS" BASIS,
* WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
* See the License for the specific language governing permissions and
* limitations under the License.
*/
/*
* To change this license header, choose License Headers in Project Properties.
* To change this template file, choose Tools | Templates
* and open the template in the editor.
*/
package org.uberfire.ext.wires.core.api.shapes;
public class UUID {
private static final char[] CHARS = "0123456789ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrs
/**
* Generate a random uuid of the specified length. Example: uuid(15) returns
* "VcydxgltxrVZSTV"
*
* @param len
*   the desired number of characters
*/
public static String uuid(int len) {
return uuid(len, CHARS.length);
UUID.java
public class UUIDGenerator {
private static final char[] CHARS = "0123456789ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmn
/**
* Generate a random uuid of the specified length. Example: uuid(15) returns
* "VcydxgltxrVZSTV"
*
* @param len
* the desired number of characters
*/
public static String uuid(int len) {
return uuid(len, CHARS.length);
}
/**
* Generate a random uuid of the specified length, and radix. Examples:
*
* uuid(8, 2) returns "01001010" (8 character ID, base=2)
* uuid(8, 10) returns "47473046" (8 character ID, base=10)
* uuid(8, 16) returns "098F4D35" (8 character ID, base=16)
*
*
* @param len
* the desired number of characters
* @param radix
* the number of allowable values for each character (must be <=
* 62)
*/
public static String uuid(int len, int radix) {
if (radix > CHARS.length) {
throw new IllegalArgumentException();
}
char[] uuid = new char[len];
// Compact form
for (int i = 0; i < len; i++) {
uuid[i] = CHARS[(int)(Math.random() * radix)];
}
return new String(uuid);
}
/**
* Generate a RFC4122, version 4 ID. Example:
* "92329D39-6F5C-4520-ABFC-AAB64544E172"
*/
public static String uuid() {
char[] uuid = new char[36];
int r;
// rfc4122 requires these characters
uuid[8] = uuid[13] = uuid[18] = uuid[23] = '-';
uuid[14] = '4';
// Fill in random data. At i==19 set the high bits of clock sequence as
// per rfc4122, sec. 4.1.5
public class UUID {
private static final char[] CHARS = "0123456789ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrs
/**
* Generate a random uuid of the specified length. Example: uuid(15) returns
* "VcydxgltxrVZSTV"
*
* @param len
*   the desired number of characters
*/
public static String uuid(int len) {
return uuid(len, CHARS.length);
}
/**
* Generate a random uuid of the specified length, and radix. Examples:
*
* uuid(8, 2) returns "01001010" (8 character ID, base=2)
* uuid(8, 10) returns "47473046" (8 character ID, base=10)
* uuid(8, 16) returns "098F4D35" (8 character ID, base=16)
*
*
* @param len
*   the desired number of characters
* @param radix
*   the number of allowable values for each character (must be <=
*   62)
*/
public static String uuid(int len, int radix) {
if (radix > CHARS.length) {
throw new IllegalArgumentException();
}
char[] uuid = new char[len];
// Compact form
for (int i = 0; i < len; i++) {
uuid[i] = CHARS[(int)(Math.random()*radix)];
}
return new String(uuid);
}
/**
* Generate a RFC4122, version 4 ID. Example:
* "92329D39-6F5C-4520-ABFC-AAB64544E172"
*/
public static String uuid() {
char[] uuid = new char[36];
int r;
// rfc4122 requires these characters
uuid[8] = uuid[13] = uuid[18] = uuid[23] = '-';
uuid[14] = '4';
// Fill in random data. At i==19 set the high bits of clock sequence as
// per rfc4122, sec. 4.1.5



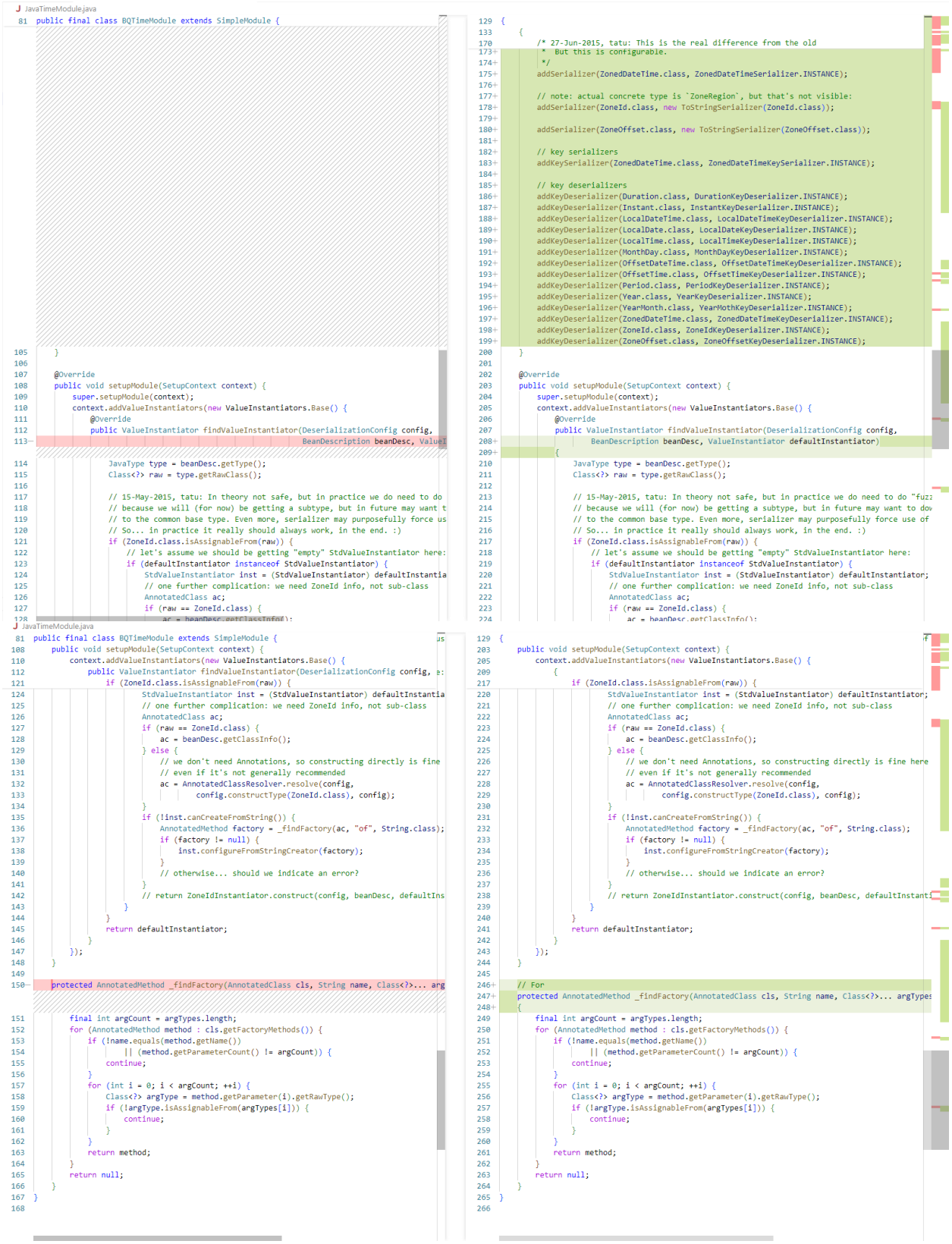
```
97      for (int i = 0; i < 36; i++) {
98          if (uuid[i] == 0) {
99              r = (int) (Math.random() * 16);
100             uuid[i] = CHARS[(i == 19) ? (r & 0x3) | 0x8 : r & 0xf];
101         }
102     }
103     return new String(uuid);
104 }
105 }
106
```



```
76      for (int i = 0; i < 36; i++) {
77          if (uuid[i] == 0) {
78              r = (int) (Math.random()*16);
79              uuid[i] = CHARS[(i == 19) ? (r & 0x3) | 0x8 : r & 0xf];
80          }
81      }
82      return new String(uuid);
83  }
84  }
85
```

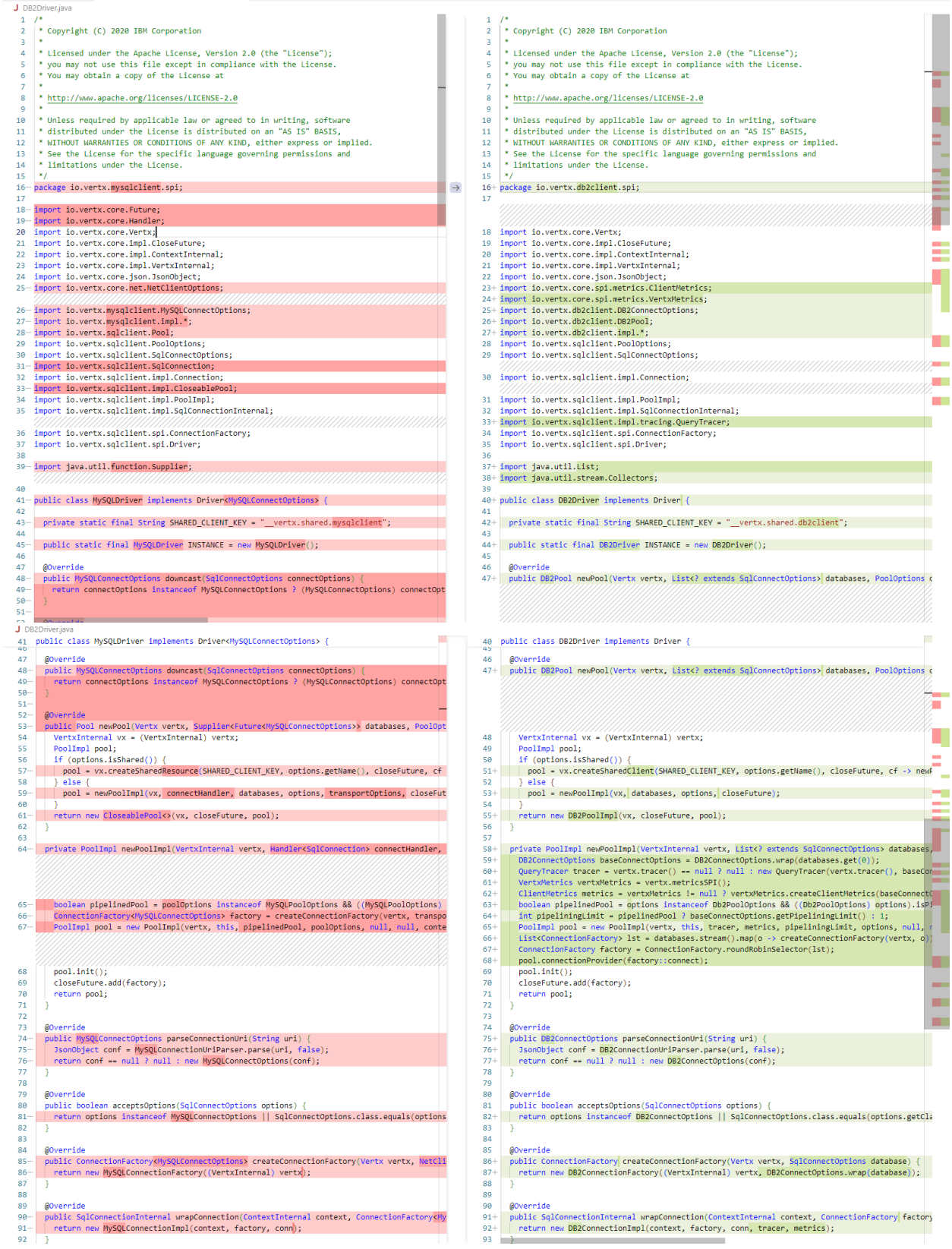
Example 2

artifact: bootique@bootique,
tpl_name: com.fasterxml.jackson.datatype:jackson-datatype-jsr310,
artifact_class:
BQTimeModule@/home/xxxx/projects/github_mvn_1k/bootique@bootique/bootique/src/main/java/io/bootique/jackson/deserializer/BQTimeModule.java,
lib_class:
JavaTimeModule@/home/xxxx/projects/mvn_critical_lib/com.fasterxml.jackson.datatype/jackson-datatype-jsr310/2.9.0.pr3/com/fasterxml/jackson/datatype/jsr310/JavaTimeModule.java,
complexity: 172.9882141524979,
pagerank: 0.21057509743915087,
percentile: 0.0,
class_id:
JavaTimeModule@/home/xxxx/projects/mvn_critical_lib/com.fasterxml.jackson.datatype/jackson-datatype-jsr310/2.9.0.pr3/com/fasterxml/jackson/datatype/jsr310/JavaTimeModule.java,
av: jackson-datatype-jsr310:2.9.0.pr3,
Comments:
Core functions setupModule and _findFactory are the same, but the constructor and license are modified.



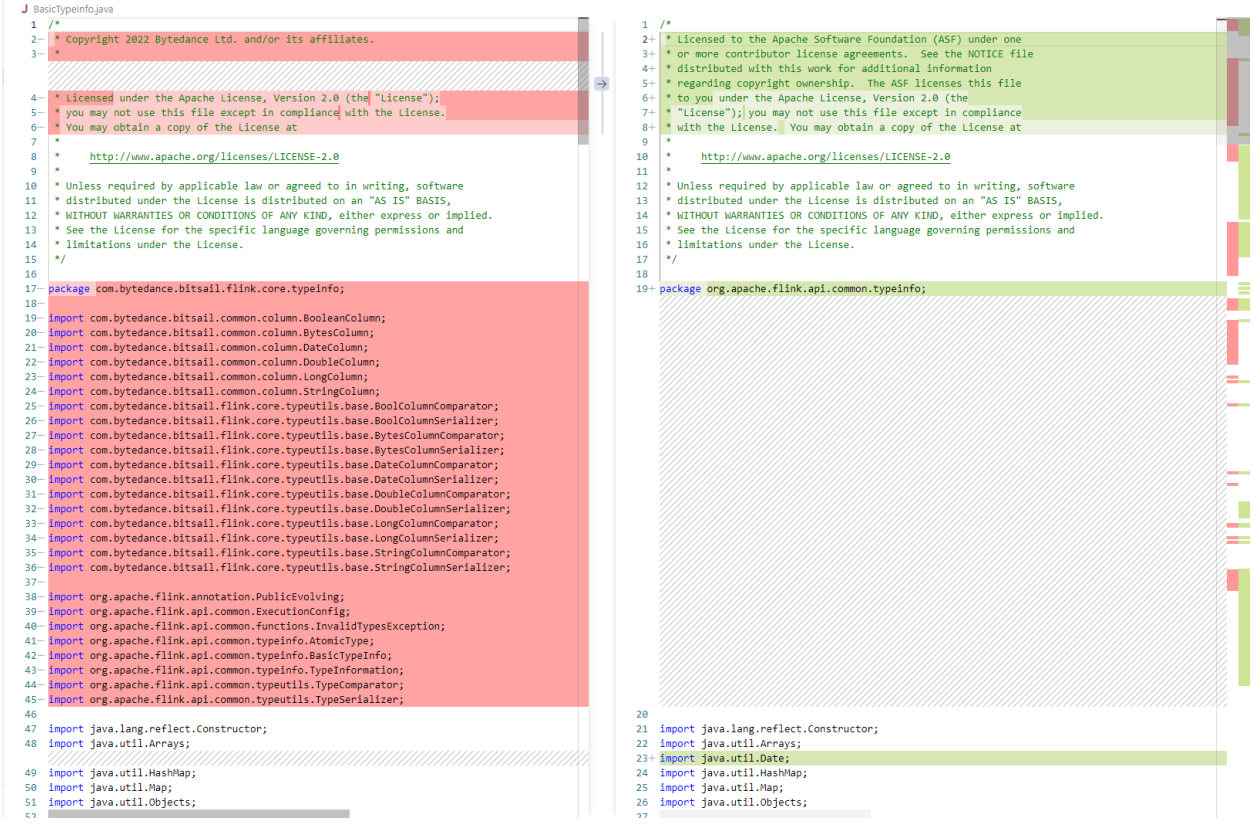
Example 3

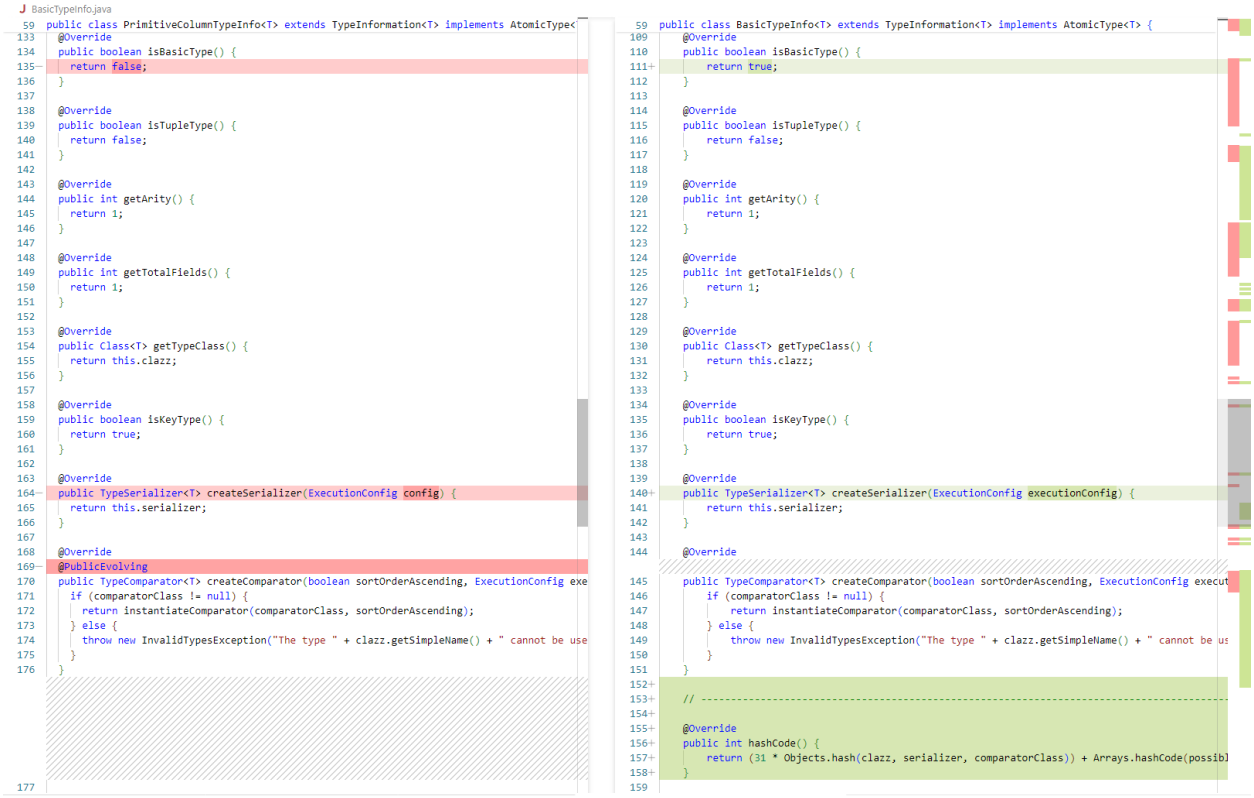
artifact: eclipse-vertx@vertx-sql-client,
tpl_name: io.vertx:vertx-db2-client,
cur_class: MySQLDriver@/home/xxxx/projects/github_mvn_1k/eclipse-vertx@vertx-sql-client/vertx-mysql-client/src/main/java/io/vertx/mysqlclient/spi/MySQLDriver.java,
lib_class: DB2Driver@/home/xxxx/projects/mvn_critical_lib/io.vertx/vertx-db2-client/4.2.5/io/vertx/db2client/spi/DB2Driver.java,
complexity: 151.38136626515677,
pagerank: 0.029308126644970383,
percentile: 0.09090909090909091,
class_id: DB2Driver@/home/xxxx/projects/mvn_critical_lib/io.vertx/vertx-db2-client/4.2.5/io/vertx/db2client/spi/DB2Driver.java,
av: vertx-db2-client:4.2.5,
Comments:
Although most variable names are replaced, basic class structures are the same. Since our method support Type 3 clone detection, this example is still detected as cloned.

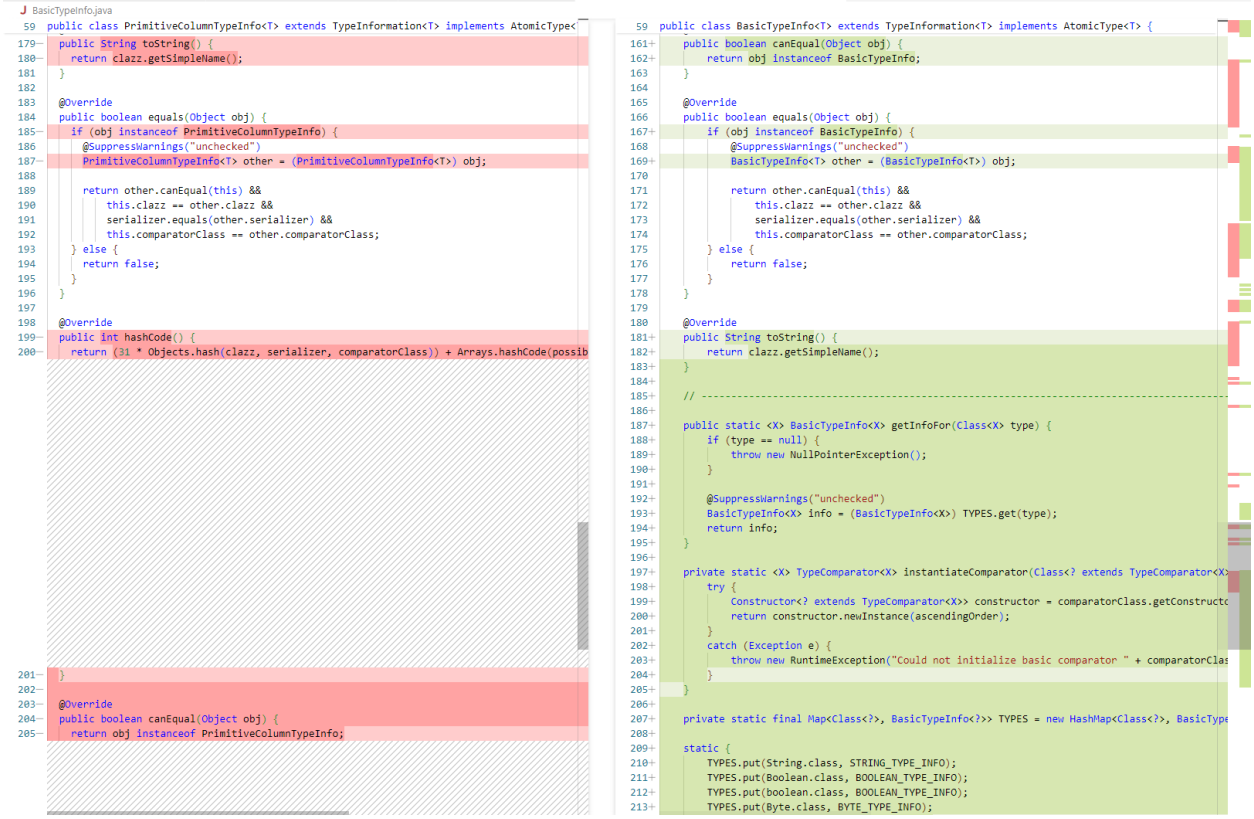


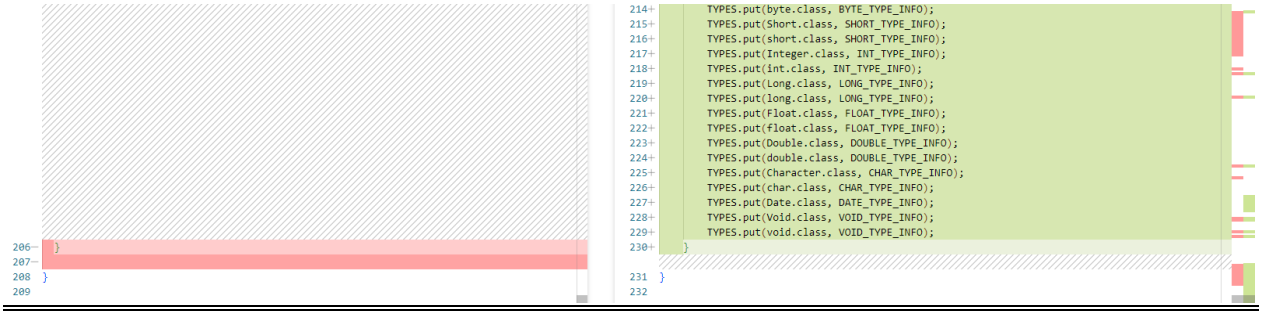
Example 4

artifact: Bytedance/bitsail ,
tpl_name: org.apache.flink:flink-core,
cur_class: PrimitiveColumnTypeInfo@/home/xxxx/projects/github_mvn_1k/bytedance@bitsail/blob/release-0.1.0/bitsail-cores/bitsail-core-flink-base/src/main/java/com/bytedance/bitsail/flink/core/typeinfo/PrimitiveColumnTypeInfo.java ,
lib_class: BasicTypeInfo@/home/xxxx/projects/mvn_critical_lib/org.apache.flink/flink-core/0.10.0/org/apache/flink/api/common/typeinfo/BasicTypeInfo.java,
complexity: 186.59134698148256,
pagerank: 0.03270665171109559,
percentile: 0.009259259259259259,
class_id: BasicTypeInfo@/home/xxxx/projects/mvn_critical_lib/org.apache.flink/flink-core/0.10.0/org/apache/flink/api/common/typeinfo/BasicTypeInfo.java,
av: flink-core:0.10.0,
Comments:
These two classes have similar structures, though their variable names are modified.



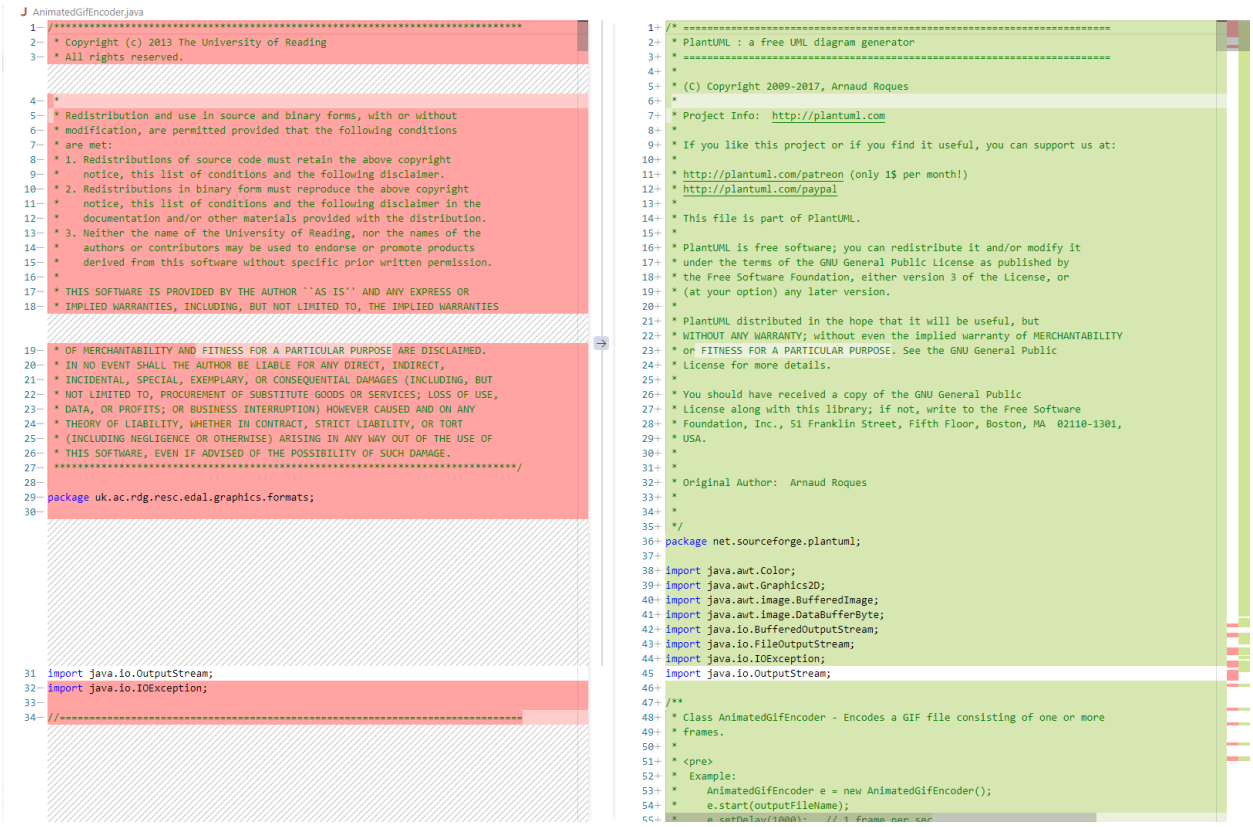

BasicTypeInfo.java
public class PrimitiveColumnTypeInfo<T> extends TypeInformation<T> implements AtomicType<
@Override
public boolean isBasicType() {
return false;
}
@Override
public boolean isTupleType() {
return false;
}
@Override
public int getArity() {
return 1;
}
@Override
public int getTotalFields() {
return 1;
}
@Override
public Class<T> getTypeClass() {
return this.clazz;
}
@Override
public boolean isKeyType() {
return true;
}
@Override
public TypeSerializer<T> createSerializer(ExecutionConfig config) {
return this.serializer;
}
@Override
@PublicEvolving
public TypeComparator<T> createComparator(boolean sortOrderAscending, ExecutionConfig exe
if (comparatorClass != null) {
return instantiateComparator(comparatorClass, sortOrderAscending);
} else {
throw new InvalidTypesException("The type " + clazz.getSimpleName() + " cannot be use
}
}
public class BasicTypeInfo<T> extends TypeInformation<T> implements AtomicType<T> {
@Override
public boolean isBasicType() {
return true;
}
@Override
public boolean isTupleType() {
return false;
}
@Override
public int getArity() {
return 1;
}
@Override
public int getTotalFields() {
return 1;
}
@Override
public Class<T> getTypeClass() {
return this.clazz;
}
@Override
public boolean isKeyType() {
return true;
}
@Override
public TypeSerializer<T> createSerializer(ExecutionConfig executionConfig) {
return this.serializer;
}
@Override
public TypeComparator<T> createComparator(boolean sortOrderAscending, ExecutionConfig execut
if (comparatorClass != null) {
return instantiateComparator(comparatorClass, sortOrderAscending);
} else {
throw new InvalidTypesException("The type " + clazz.getSimpleName() + " cannot be us
}
}
// --
@Override
public int hashCode() {
return (31 * Objects.hash(clazz, serializer, comparatorClass)) + Arrays.hashCode(possibl
}


BasicTypeInfo.java
public class PrimitiveColumnTypeInfo<T> extends TypeInformation<T> implements AtomicType<
public String toString() {
return clazz.getSimpleName();
}
@Override
public boolean equals(Object obj) {
if (obj instanceof PrimitiveColumnTypeInfo) {
@SuppressWarnings("unchecked")
PrimitiveColumnTypeInfo<T> other = (PrimitiveColumnTypeInfo<T>) obj;
return other.canEqual(this) &&
this.clazz == other.clazz &&
serializer.equals(other.serializer) &&
this.comparatorClass == other.comparatorClass;
} else {
return false;
}
}
@Override
public int hashCode() {
return (31 * Objects.hash(clazz, serializer, comparatorClass)) + Arrays.hashCode(possib
}
@Override
public boolean canEqual(Object obj) {
return obj instanceof PrimitiveColumnTypeInfo;
public class BasicTypeInfo<T> extends TypeInformation<T> implements AtomicType<T> {
public boolean canEqual(Object obj) {
return obj instanceof BasicTypeInfo;
}
@Override
public boolean equals(Object obj) {
if (obj instanceof BasicTypeInfo) {
@SuppressWarnings("unchecked")
BasicTypeInfo<T> other = (BasicTypeInfo<T>) obj;
return other.canEqual(this) &&
this.clazz == other.clazz &&
serializer.equals(other.serializer) &&
this.comparatorClass == other.comparatorClass;
} else {
return false;
}
}
@Override
public String toString() {
return clazz.getSimpleName();
}
// --
public static <X> BasicTypeInfo<X> getInfoFor(Class<X> type) {
if (type == null) {
throw new NullPointerException();
}
@SuppressWarnings("unchecked")
BasicTypeInfo<X> info = (BasicTypeInfo<X>) TYPES.get(type);
return info;
}
private static <X> TypeComparator<X> instantiateComparator(Class<? extends TypeComparator<X>
try {
Constructor<? extends TypeComparator<X>> constructor = comparatorClass.getConstructo
return constructor.newInstance(ascendingOrder);
}
catch (Exception e) {
throw new RuntimeException("Could not initialize basic comparator " + comparatorClas
}
}
private static final Map<Class<?>, BasicTypeInfo<?>> TYPES = new HashMap<Class<?>, BasicType
static {
TYPES.put(String.class, STRING_TYPE_INFO);
TYPES.put(Boolean.class, BOOLEAN_TYPE_INFO);
TYPES.put(boolean.class, BOOLEAN_TYPE_INFO);
TYPES.put(Byte.class, BYTE_TYPE_INFO);

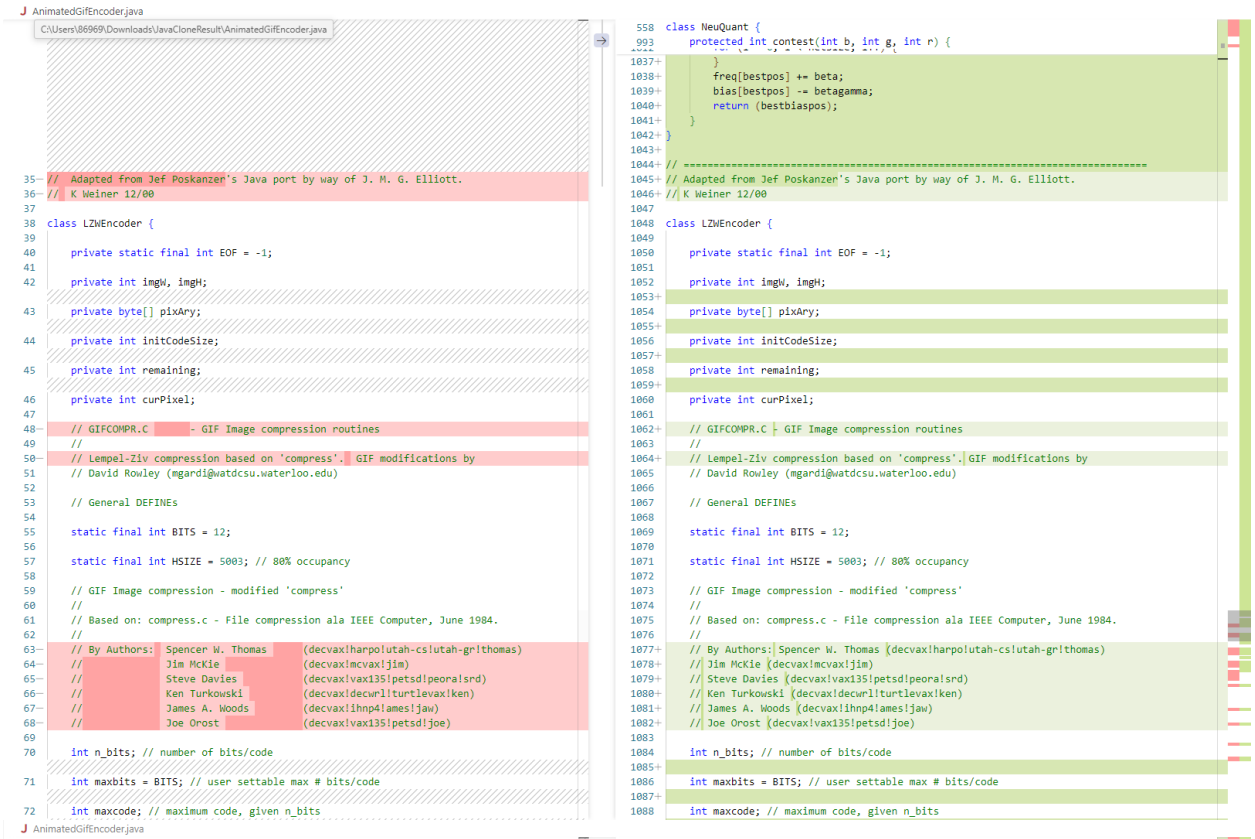


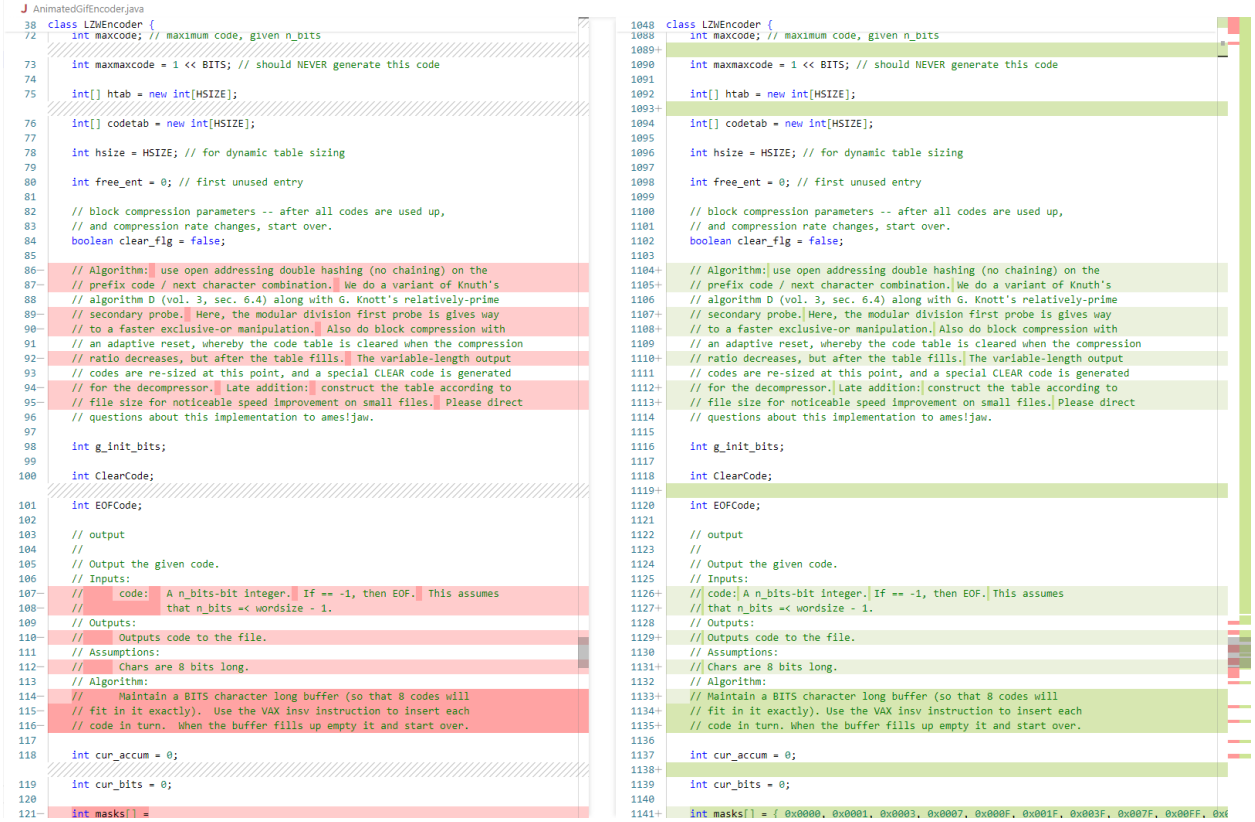
Example 5

artifact: Reading-eScience-Centre@edal-java,
tpl_name: net.sourceforge.plantuml:plantuml,
cur_class: LZWEncoder@/home/xxxx/projects/github_mvn_1k/Reading-eScience-Centre@edal-java/graphics/src/main/java/uk/ac/rdg/resc/edal/graphics/formats/LZWEncoder.java,
lib_class: LZWEncoder@/home/xxxx/projects/mvn_critical_lib/net.sourceforge.plantuml/plantuml/2017.09/net/sourceforge/plantuml/AnimatedGifEncoder.java,
complexity: 200.08224021189676,
pagerank: 9.070746570380477e-05,
percentile: 0.4817315329626688,
class_id: LZWEncoder@/home/xxxx/projects/mvn_critical_lib/net.sourceforge.plantuml/plantuml/2017.09/net/sourceforge/plantuml/AnimatedGifEncoder.java,
av: plantuml:2017.09,
Comments:
This example was detailed in the paper that only part of the files (i.e., class LZWEncoder) is reused, and another part (i.e., class AnimatedGifEncoder) is not copied at all.



<Omitting 1000 lines of class AnimatedGifEncoder on the right side, which is not copied or related to the file on the left side.>


AnimatedGifEncoder.java
C:\Users\86969\Downloads\JavaCloneResult\AnimatedGifEncoder.java
// Adapted from Jef Poskanzer's Java port by way of J. M. G. Elliott.
// K Weiner 12/00
class LZWEncoder {
private static final int EOF = -1;
private int imgW, imgH;
private byte[] pixAry;
private int initCodeSize;
private int remaining;
private int curPixel;
// GIFCOMPR.C - GIF Image compression routines
// Lempel-Ziv compression based on 'compress'. GIF modifications by
// David Rowley (mgardi@watdcsu.waterloo.edu)
// General DEFINEs
static final int BITS = 12;
static final int HSIZE = 5003; // 80% occupancy
// GIF Image compression - modified 'compress'
// Based on: compress.c - File compression ala IEEE Computer, June 1984.
// By Authors: Spencer W. Thomas (decvax!harpo!utah-cs!utah-gr!thomas)
// Jim McKie (decvax!mcvax!jim)
// Steve Davies (decvax!vax135!petsd!peora!srd)
// Ken Turkowski (decvax!decwrl!turtlevax!ken)
// James A. Woods (decvax!ihnp4!ames!jaw)
// Joe Orost (decvax!vax135!petsd!joe)
int n_bits; // number of bits/code
int maxbits = BITS; // user settable max # bits/code
int maxcode; // maximum code, given n_bits
class NeuQuant {
protected int contest(int b, int g, int r) {
}
freq[bestpos] += beta;
bias[bestpos] -= betagamma;
return (bestbiaspos);
}
}
// ==========
// Adapted from Jef Poskanzer's Java port by way of J. M. G. Elliott.
// K Weiner 12/00
class LZWEncoder {
private static final int EOF = -1;
private int imgW, imgH;
private byte[] pixAry;
private int initCodeSize;
private int remaining;
private int curPixel;
// GIFCOMPR.C - GIF Image compression routines
// Lempel-Ziv compression based on 'compress'. GIF modifications by
// David Rowley (mgardi@watdcsu.waterloo.edu)
// General DEFINEs
static final int BITS = 12;
static final int HSIZE = 5003; // 80% occupancy
// GIF Image compression - modified 'compress'
// Based on: compress.c - File compression ala IEEE Computer, June 1984.
// By Authors: Spencer W. Thomas (decvax!harpo!utah-cs!utah-gr!thomas)
// Jim McKie (decvax!mcvax!jim)
// Steve Davies (decvax!vax135!petsd!peora!srd)
// Ken Turkowski (decvax!decwrl!turtlevax!ken)
// James A. Woods (decvax!ihnp4!ames!jaw)
// Joe Orost (decvax!vax135!petsd!joe)
int n_bits; // number of bits/code
int maxbits = BITS; // user settable max # bits/code
int maxcode; // maximum code, given n_bits


AnimatedGifEncoder.java
class LZWEncoder {
int maxcode; // maximum code, given n_bits
int maxmaxcode = 1 << BITS; // should NEVER generate this code
int[] htab = new int[HSIZE];
int[] codetab = new int[HSIZE];
int hsize = HSIZE; // for dynamic table sizing
int free_ent = 0; // first unused entry
// block compression parameters -- after all codes are used up,
// and compression rate changes, start over.
boolean clear_flg = false;
// Algorithm: use open addressing double hashing (no chaining) on the
// prefix code / next character combination. We do a variant of Knuth's
// algorithm D (vol. 3, sec. 6.4) along with G. Knott's relatively-prime
// secondary probe. Here, the modular division first probe is gives way
// to a faster exclusive-or manipulation. Also do block compression with
// an adaptive reset, whereby the code table is cleared when the compression
// ratio decreases, but after the table fills. The variable-length output
// codes are re-sized at this point, and a special CLEAR code is generated
// for the decompressor. Late addition: construct the table according to
// file size for noticeable speed improvement on small files. Please direct
// questions about this implementation to ames!jaw.
int g_init_bits;
int ClearCode;
int EOFCode;
// output
// Output the given code.
// Inputs:
// code: A n_bits-bit integer. If == -1, then EOF. This assumes
// that n_bits =< wordsize - 1.
// Outputs:
// Outputs code to the file.
// Assumptions:
// Chars are 8 bits long.
// Algorithm:
// Maintain a BITS character long buffer (so that 8 codes will
// fit in it exactly). Use the VAX insv instruction to insert each
// code in turn. When the buffer fills up empty it and start over.
int cur_accum = 0;
int cur_bits = 0;
int masks[] =
class LZWEncoder {
int maxcode; // maximum code, given n_bits
int maxmaxcode = 1 << BITS; // should NEVER generate this code
int[] htab = new int[HSIZE];
int[] codetab = new int[HSIZE];
int hsize = HSIZE; // for dynamic table sizing
int free_ent = 0; // first unused entry
// block compression parameters -- after all codes are used up,
// and compression rate changes, start over.
boolean clear_flg = false;
// Algorithm: use open addressing double hashing (no chaining) on the
// prefix code / next character combination. We do a variant of Knuth's
// algorithm D (vol. 3, sec. 6.4) along with G. Knott's relatively-prime
// secondary probe. Here, the modular division first probe is gives way
// to a faster exclusive-or manipulation. Also do block compression with
// an adaptive reset, whereby the code table is cleared when the compression
// ratio decreases, but after the table fills. The variable-length output
// codes are re-sized at this point, and a special CLEAR code is generated
// for the decompressor. Late addition: construct the table according to
// file size for noticeable speed improvement on small files. Please direct
// questions about this implementation to ames!jaw.
int g_init_bits;
int ClearCode;
int EOFCode;
// output
// Output the given code.
// Inputs:
// code: A n_bits-bit integer. If == -1, then EOF. This assumes
// that n_bits =< wordsize - 1.
// Outputs:
// Outputs code to the file.
// Assumptions:
// Chars are 8 bits long.
// Algorithm:
// Maintain a BITS character long buffer (so that 8 codes will
// fit in it exactly). Use the VAX insv instruction to insert each
// code in turn. When the buffer fills up empty it and start over.
int cur_accum = 0;
int cur_bits = 0;
int masks[] = { 0x0000, 0x0001, 0x0003, 0x0007, 0x000F, 0x001F, 0x003F, 0x007F, 0x00FF, 0x

AnimatedGifEncoder.java

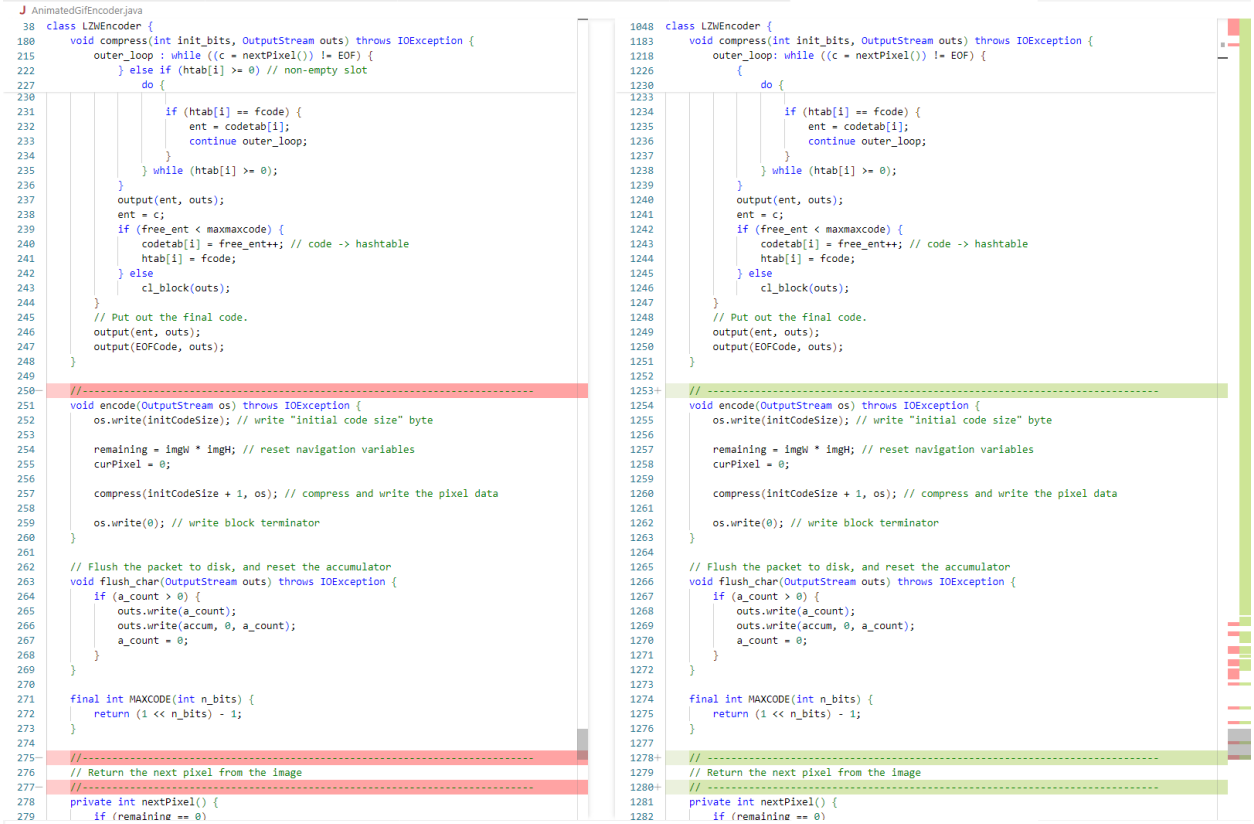
```
38  class LZWEncoder {
121-    int masks[] =
122-    {
123-        0x0000,
124-        0x0001,
125-        0x0003,
126-        0x0007,
127-        0x000F,
128-        0x001F,
129-        0x003F,
130-        0x007F,
131-        0x00FF,
132-        0x01FF,
133-        0x03FF,
134-        0x07FF,
135-        0x0FFF,
136-        0x1FFF,
137-        0x3FFF,
138-        0x7FFF,
139-        0xFFFF };
140
141     // Number of characters so far in this 'packet'
142     int a_count;
143
144     // Define the storage for the packet accumulator
145     byte[] accum = new byte[256];
146
147-    //------------------------------------------------------------------------
148     LZWEncoder(int width, int height, byte[] pixels, int color_depth) {
149         imgW = width;
150         imgH = height;
151         pixAry = pixels;
152         initCodeSize = Math.max(2, color_depth);
153     }
154
155     // Add a character to the end of the current packet, and if it is 254
156     // characters, flush the packet to disk.
157     void char_out(byte c, OutputStream outs) throws IOException {
158         accum[a_count++] = c;
159         if (a_count >= 254)
160             flush_char(outs);
161     }
162
163     // Clear out the hash table
164
165     // table clear for block compress
166     void cl_block(OutputStream outs) throws IOException {
167         cl_hash(hsize);
168         free_ent = ClearCode + 2;
169         clear_flg = true;
170
171         output(ClearCode, outs);
172     }
173
```

```
1048  class LZWEncoder {
1141+     int masks[] = { 0x0000, 0x0001, 0x0003, 0x0007, 0x000F, 0x001F, 0x003F, 0x007F, 0x00FF, 0x
1142+         0x0FFF, 0x1FFF, 0x3FFF, 0x7FFF, 0xFFFF };
1143
1144      // Number of characters so far in this 'packet'
1145      int a_count;
1146
1147      // Define the storage for the packet accumulator
1148      byte[] accum = new byte[256];
1149
1150+     //------------------------------------------------------------------------
1151      LZWEncoder(int width, int height, byte[] pixels, int color_depth) {
1152          imgW = width;
1153          imgH = height;
1154          pixAry = pixels;
1155          initCodeSize = Math.max(2, color_depth);
1156      }
1157
1158      // Add a character to the end of the current packet, and if it is 254
1159      // characters, flush the packet to disk.
1160      void char_out(byte c, OutputStream outs) throws IOException {
1161          accum[a_count++] = c;
1162          if (a_count >= 254)
1163              flush_char(outs);
1164      }
1165
1166      // Clear out the hash table
1167
1168      // table clear for block compress
1169      void cl_block(OutputStream outs) throws IOException {
1170          cl_hash(hsize);
1171          free_ent = ClearCode + 2;
1172          clear_flg = true;
1173
1174          output(ClearCode, outs);
1175      }
1176
```

AnimatedGifEncoder.java

```
38  class LZWEncoder {
174     // reset code table
175     void cl_hash(int hsize) {
176         for (int i = 0; i < hsize; ++i)
177             htab[i] = -1;
178     }
179
180     void compress(int init_bits, OutputStream outs) throws IOException {
181         int fcode;
182         int i /* = 0 */;
183         int c;
184         int ent;
185         int disp;
186         int hsize_reg;
187         int hshift;
188
189-        // Set up the globals:  g_init_bits - initial number of bits
190         g_init_bits = init_bits;
191
192         // Set up the necessary values
193         clear_flg = false;
194         n_bits = g_init_bits;
195         maxcode = MAXCODE(n_bits);
196
197         ClearCode = 1 << (init_bits - 1);
198         EOFCode = ClearCode + 1;
199         free_ent = ClearCode + 2;
200
201         a_count = 0; // clear packet
202
203         ent = nextPixel();
204
205         hshift = 0;
206         for (fcode = hsize; fcode < 65536; fcode *= 2)
207             ++hshift;
208         hshift = 8 - hshift; // set hash code range bound
209
210         hsize_reg = hsize;
211         cl_hash(hsize_reg); // clear hash table
212
213         output(ClearCode, outs);
214
215-        outer_loop : while ((c = nextPixel()) != EOF) {
216             fcode = (c << maxbits) + ent;
217             i = (c << hshift) ^ ent; // xor hashing
218
219             if (htab[i] == fcode) {
220                 ent = codetab[i];
221                 continue;
222             } else if (htab[i] >= 0) // non-empty slot
223                 {
224                 disp = hsize_reg - i; // secondary hash (after G. Knott)
225                 if (i == 0)
226                     disp = 1;
```

```
1048  class LZWEncoder {
1177      // reset code table
1178      void cl_hash(int hsize) {
1179          for (int i = 0; i < hsize; ++i)
1180              htab[i] = -1;
1181      }
1182
1183      void compress(int init_bits, OutputStream outs) throws IOException {
1184          int fcode;
1185          int i /* = 0 */;
1186          int c;
1187          int ent;
1188          int disp;
1189          int hsize_reg;
1190          int hshift;
1191
1192+         // Set up the globals: g_init_bits - initial number of bits
1193          g_init_bits = init_bits;
1194
1195          // Set up the necessary values
1196          clear_flg = false;
1197          n_bits = g_init_bits;
1198          maxcode = MAXCODE(n_bits);
1199
1200          ClearCode = 1 << (init_bits - 1);
1201          EOFCode = ClearCode + 1;
1202          free_ent = ClearCode + 2;
1203
1204          a_count = 0; // clear packet
1205
1206          ent = nextPixel();
1207
1208          hshift = 0;
1209          for (fcode = hsize; fcode < 65536; fcode *= 2)
1210              ++hshift;
1211          hshift = 8 - hshift; // set hash code range bound
1212
1213          hsize_reg = hsize;
1214          cl_hash(hsize_reg); // clear hash table
1215
1216          output(ClearCode, outs);
1217
1218+         outer_loop: while ((c = nextPixel()) != EOF) {
1219              fcode = (c << maxbits) + ent;
1220              i = (c << hshift) ^ ent; // xor hashing
1221
1222              if (htab[i] == fcode) {
1223                  ent = codetab[i];
1224                  continue;
1225              } else if (htab[i] >= 0) // non-empty slot
1226              {
1227                  disp = hsize_reg - i; // secondary hash (after G. Knott)
1228                  if (i == 0)
1229                      disp = 1;
```

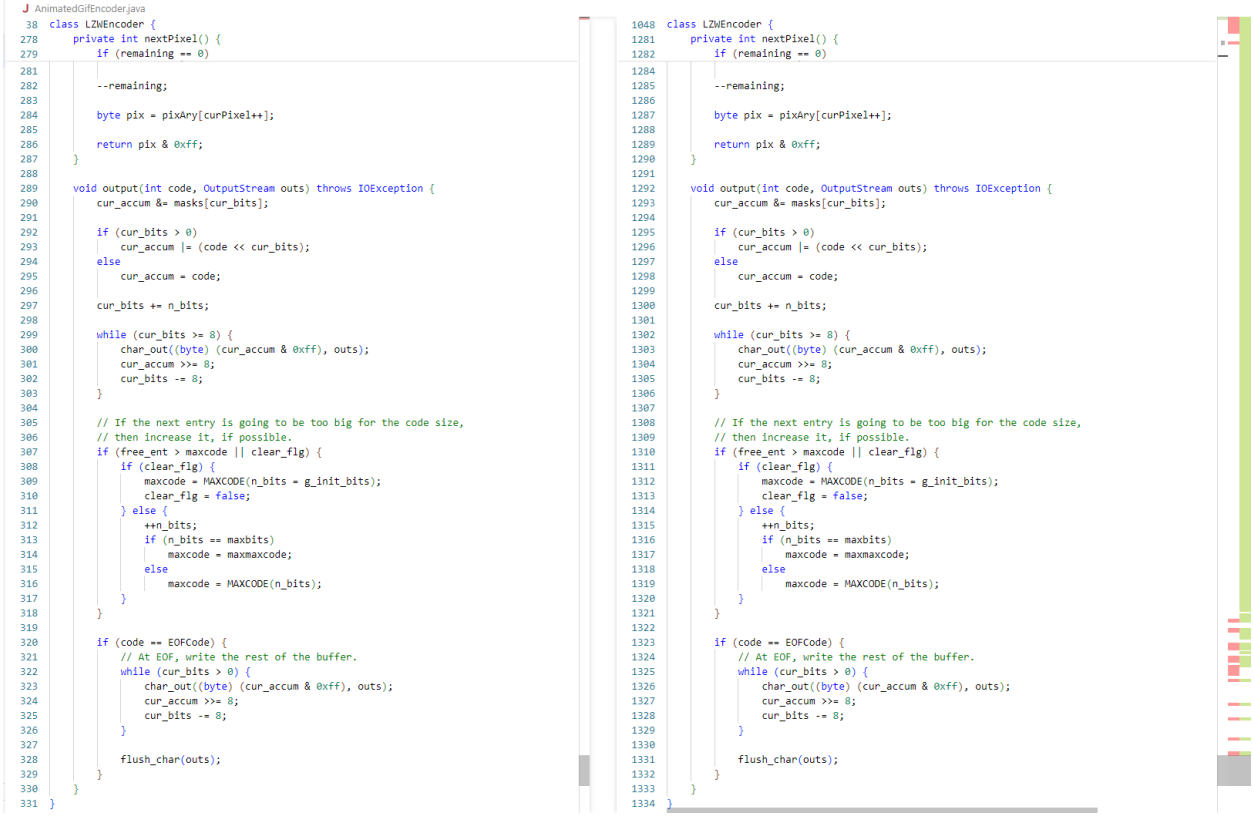


AnimatedGifEncoder.java

```
class LZWEncoder {
    void compress(int init_bits, OutputStream outs) throws IOException {
        outer_loop : while ((c = nextPixel()) != EOF) {
            } else if (htab[i] >= 0) // non-empty slot
                do {
                    if (htab[i] == fcode) {
                        ent = codetab[i];
                        continue outer_loop;
                    }
                } while (htab[i] >= 0);
            }
            output(ent, outs);
            ent = c;
            if (free_ent < maxmaxcode) {
                codetab[i] = free_ent++; // code -> hashtable
                htab[i] = fcode;
            } else
                cl_block(outs);
        }
        // Put out the final code.
        output(ent, outs);
        output(EOFCode, outs);
    }

    //------------------------------------------------------------------------
    void encode(OutputStream os) throws IOException {
        os.write(initCodeSize); // write "initial code size" byte

        remaining = imgW * imgH; // reset navigation variables
        curPixel = 0;

        compress(initCodeSize + 1, os); // compress and write the pixel data

        os.write(0); // write block terminator
    }

    // Flush the packet to disk, and reset the accumulator
    void flush_char(OutputStream outs) throws IOException {
        if (a_count > 0) {
            outs.write(a_count);
            outs.write(accum, 0, a_count);
            a_count = 0;
        }
    }

    final int MAXCODE(int n_bits) {
        return (1 << n_bits) - 1;
    }

    //------------------------------------------------------------------------
    // Return the next pixel from the image
    //------------------------------------------------------------------------
    private int nextPixel() {
        if (remaining == 0)
```


AnimatedGifEncoder.java

```
class LZWEncoder {
    void compress(int init_bits, OutputStream outs) throws IOException {
        outer_loop: while ((c = nextPixel()) != EOF) {
            {
                do {
                    if (htab[i] == fcode) {
                        ent = codetab[i];
                        continue outer_loop;
                    }
                } while (htab[i] >= 0);
            }
            output(ent, outs);
            ent = c;
            if (free_ent < maxmaxcode) {
                codetab[i] = free_ent++; // code -> hashtable
                htab[i] = fcode;
            } else
                cl_block(outs);
        }
        // Put out the final code.
        output(ent, outs);
        output(EOFCode, outs);
    }

    // ------------------------------------------------------------------------
    void encode(OutputStream os) throws IOException {
        os.write(initCodeSize); // write "initial code size" byte

        remaining = imgW * imgH; // reset navigation variables
        curPixel = 0;

        compress(initCodeSize + 1, os); // compress and write the pixel data

        os.write(0); // write block terminator
    }

    // Flush the packet to disk, and reset the accumulator
    void flush_char(OutputStream outs) throws IOException {
        if (a_count > 0) {
            outs.write(a_count);
            outs.write(accum, 0, a_count);
            a_count = 0;
        }
    }

    final int MAXCODE(int n_bits) {
        return (1 << n_bits) - 1;
    }

    // ------------------------------------------------------------------------
    // Return the next pixel from the image
    // ------------------------------------------------------------------------
    private int nextPixel() {
        if (remaining == 0)
```



AnimatedGifEncoder.java

```
class LZWEncoder {
    private int nextPixel() {
        if (remaining == 0)

        --remaining;

        byte pix = pixAry[curPixel++];

        return pix & 0xff;
    }

    void output(int code, OutputStream outs) throws IOException {
        cur_accum &= masks[cur_bits];

        if (cur_bits > 0)
            cur_accum |= (code << cur_bits);
        else
            cur_accum = code;

        cur_bits += n_bits;

        while (cur_bits >= 8) {
            char_out((byte) (cur_accum & 0xff), outs);
            cur_accum >>= 8;
            cur_bits -= 8;
        }

        // If the next entry is going to be too big for the code size,
        // then increase it, if possible.
        if (free_ent > maxcode || clear_flg) {
            if (clear_flg) {
                maxcode = MAXCODE(n_bits = g_init_bits);
                clear_flg = false;
            } else {
                ++n_bits;
                if (n_bits == maxbits)
                    maxcode = maxmaxcode;
                else
                    maxcode = MAXCODE(n_bits);
            }
        }

        if (code == EOFCode) {
            // At EOF, write the rest of the buffer.
            while (cur_bits > 0) {
                char_out((byte) (cur_accum & 0xff), outs);
                cur_accum >>= 8;
                cur_bits -= 8;
            }

            flush_char(outs);
        }
    }
}
```


AnimatedGifEncoder.java

```
class LZWEncoder {
    private int nextPixel() {
        if (remaining == 0)

        --remaining;

        byte pix = pixAry[curPixel++];

        return pix & 0xff;
    }

    void output(int code, OutputStream outs) throws IOException {
        cur_accum &= masks[cur_bits];

        if (cur_bits > 0)
            cur_accum |= (code << cur_bits);
        else
            cur_accum = code;

        cur_bits += n_bits;

        while (cur_bits >= 8) {
            char_out((byte) (cur_accum & 0xff), outs);
            cur_accum >>= 8;
            cur_bits -= 8;
        }

        // If the next entry is going to be too big for the code size,
        // then increase it, if possible.
        if (free_ent > maxcode || clear_flg) {
            if (clear_flg) {
                maxcode = MAXCODE(n_bits = g_init_bits);
                clear_flg = false;
            } else {
                ++n_bits;
                if (n_bits == maxbits)
                    maxcode = maxmaxcode;
                else
                    maxcode = MAXCODE(n_bits);
            }
        }

        if (code == EOFCode) {
            // At EOF, write the rest of the buffer.
            while (cur_bits > 0) {
                char_out((byte) (cur_accum & 0xff), outs);
                cur_accum >>= 8;
                cur_bits -= 8;
            }

            flush_char(outs);
        }
    }
}
```