

## 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 version
- **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.



```

97     for (int i = 0; i < 36; i++) {
98         if (uuid[i] == 0) {
99             r = (int) (Math.random() * 16);
100             uuid[i] = CHARs[(1 == 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[(1 == 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.

J JavaTimeModule.java

```
1  /*
2  * Licensed to ObjectStyle LLC under one
3  * or more contributor license agreements. See the NOTICE file
4  * distributed with this work for additional information
5  * regarding copyright ownership. The ObjectStyle LLC licenses
6  * this file to you under the Apache License, Version 2.0 (the
7  * "License"); you may not use this file except in compliance
8  * with the License. You may obtain a copy of the License at
9  *
10 * http://www.apache.org/licenses/LICENSE-2.0
11 *
12 * Unless required by applicable law or agreed to in writing,
13 * software distributed under the License is distributed on an
14 * "AS IS" BASIS, WITHOUT WARRANTIES OR CONDITIONS OF ANY
15 * KIND, either express or implied. See the License for the
16 * specific language governing permissions and limitations
17 * under the License.
18 */
19
20 package io.bootique.jackson.deserializer;
21
22 import com.fasterxml.jackson.core.json.PackageVersion;
23 import com.fasterxml.jackson.databind.BeanDescription;
24 import com.fasterxml.jackson.databind.DeserializationConfig;
25 import com.fasterxml.jackson.databind.JavaType;
26 import com.fasterxml.jackson.databind.deser.ValueInstantiator;
27 import com.fasterxml.jackson.databind.deser.ValueInstantiators;
28 import com.fasterxml.jackson.databind.deser.std.StdValueInstantiator;
29 import com.fasterxml.jackson.databind.introspect.AnnotatedClass;
30 import com.fasterxml.jackson.databind.introspect.AnnotatedClassResolver;
31 import com.fasterxml.jackson.databind.introspect.AnnotatedMethod;
32 import com.fasterxml.jackson.databind.module.SimpleModule;
33
34 import java.time.Duration;
35 import java.time.Instant;
36 import java.time.LocalDate;
37 import java.time.LocalDateTime;
38 import java.time.LocalTime;
39 import java.time.MonthDay;
40 import java.time.OffsetDateTime;
41 import java.time.OffsetTime;
42 import java.time.Period;
43 import java.time.Year;
44 import java.time.YearMonth;
45 import java.time.ZoneId;
46 import java.time.ZoneOffset;
47 import java.time.ZonedDateTime;
48
49 /**
```

J JavaTimeModule.java

```
77 * and are instead represented as arrays when WRITE_DATES_AS_TIMESTAMPS is enabled.</li>
78 * </ul>
79 */
80 @SuppressWarnings("javadoc")
81 public final class BQTimeModule extends SimpleModule {
82
83     public BQTimeModule() {
84
85         super(PackageVersion.VERSION);
86
87         // First deserializers
88
89         // // Instant variants:
90         addDeserializer(Instant.class, InstantDeserializer.INSTANT);
91         addDeserializer(OffsetDateTime.class, InstantDeserializer.OFFSET_DATE_TIME);
92         addDeserializer(ZonedDateTime.class, InstantDeserializer.ZONED_DATE_TIME);
93
94         // // Other deserializers
95         addDeserializer(Duration.class, DurationDeserializer.INSTANT);
96         addDeserializer(LocalDateTime.class, LocalDateTimeDeserializer.INSTANT);
97         addDeserializer(LocalDate.class, LocalDateDeserializer.INSTANT);
98         addDeserializer(LocalTime.class, LocalTimeDeserializer.INSTANT);
99         addDeserializer(MonthDay.class, JSR310StringParsableDeserializer.MONTH_DAY);
100         addDeserializer(OffsetTime.class, OffsetTimeDeserializer.INSTANT);
101         addDeserializer(Period.class, JSR310StringParsableDeserializer.PERIOD);
102         addDeserializer(Year.class, YearDeserializer.INSTANT);
103         addDeserializer(YearMonth.class, YearMonthDeserializer.INSTANT);
104         addDeserializer(ZoneId.class, JSR310StringParsableDeserializer.ZONE_ID);
105         addDeserializer(ZoneOffset.class, JSR310StringParsableDeserializer.ZONE_OFFSET);
106     }
107 }
```

```
1  /*
2  * Copyright 2013 FasterXML.com
3  *
4  * Licensed under the Apache License, Version 2.0 (the "License"); you may
5  * not use this file except in compliance with the License. You may obtain
6  * a copy of the License at
7  *
8  * http://www.apache.org/licenses/LICENSE-2.0
9  *
10 * Unless required by applicable law or agreed to in writing, software
11 * distributed under the License is distributed on an "AS IS" BASIS,
12 * WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
13 * See the license for the specific language governing permissions and
14 * limitations under the license.
15 */
16
17 package com.fasterxml.jackson.datatype.jsr310;
18
19 import java.time.Duration;
20 import java.time.Instant;
21 import java.time.LocalDate;
22 import java.time.LocalDateTime;
23 import java.time.LocalTime;
24 import java.time.MonthDay;
25 import java.time.OffsetDateTime;
26 import java.time.OffsetTime;
27 import java.time.Period;
28 import java.time.Year;
29 import java.time.YearMonth;
30 import java.time.ZoneId;
31 import java.time.ZoneOffset;
32 import java.time.ZonedDateTime;
33
34 import com.fasterxml.jackson.databind.BeanDescription;
35 import com.fasterxml.jackson.databind.DeserializationConfig;
36 import com.fasterxml.jackson.databind.JavaType;
37 import com.fasterxml.jackson.databind.deser.ValueInstantiator;
38 import com.fasterxml.jackson.databind.deser.ValueInstantiators;
39 import com.fasterxml.jackson.databind.deser.std.StdValueInstantiator;
40 import com.fasterxml.jackson.databind.introspect.AnnotatedClass
```

```
82 /**
83 * and are instead represented as arrays when WRITE_DATES_AS_TIMESTAMPS is enabled.</li>
84 * </ul>
85 */
86 @author Nick Williams
87 @author Zoltan Kiss
88 @since 2.6.0
89 @see com.fasterxml.jackson.datatype.jsr310.ser.key.Jsr310NullKeySerializer
90 */
91 @SuppressWarnings("javadoc")
92 public final class JavaTimeModule extends SimpleModule
93 {
94     private static final long serialVersionUID = 1L;
95
96     public JavaTimeModule() {
97
98         super(PackageVersion.VERSION);
99
100         // First deserializers
101
102         // // Instant variants:
103         addDeserializer(Instant.class, InstantDeserializer.INSTANT);
104         addDeserializer(OffsetDateTime.class, InstantDeserializer.OFFSET_DATE_TIME);
105         addDeserializer(ZonedDateTime.class, InstantDeserializer.ZONED_DATE_TIME);
106
107         // // Other deserializers
108         addDeserializer(Duration.class, DurationDeserializer.INSTANT);
109         addDeserializer(LocalDateTime.class, LocalDateTimeDeserializer.INSTANT);
110         addDeserializer(LocalDate.class, LocalDateDeserializer.INSTANT);
111         addDeserializer(LocalTime.class, LocalTimeDeserializer.INSTANT);
112         addDeserializer(MonthDay.class, MonthDayDeserializer.INSTANT);
113         addDeserializer(OffsetTime.class, OffsetTimeDeserializer.INSTANT);
114         addDeserializer(Period.class, JSR310StringParsableDeserializer.PERIOD);
115         addDeserializer(Year.class, YearDeserializer.INSTANT);
116         addDeserializer(YearMonth.class, YearMonthDeserializer.INSTANT);
117         addDeserializer(ZoneId.class, JSR310StringParsableDeserializer.ZONE_ID);
118         addDeserializer(ZoneOffset.class, JSR310StringParsableDeserializer.ZONE_OFFSET);
119
120         // then serializers:
121         addSerializer(Duration.class, DurationSerializer.INSTANT);
122         addSerializer(Instant.class, InstantSerializer.INSTANT);
123         addSerializer(LocalDateTime.class, LocalDateTimeSerializer.INSTANT);
124         addSerializer(LocalDate.class, LocalDateSerializer.INSTANT);
125         addSerializer(LocalTime.class, LocalTimeSerializer.INSTANT);
126         addSerializer(MonthDay.class, MonthDaySerializer.INSTANT);
127         addSerializer(OffsetDateTime.class, OffsetDateTimeSerializer.INSTANT);
128         addSerializer(OffsetTime.class, OffsetTimeSerializer.INSTANT);
129         addSerializer(Period.class, new ToStringSerializer(Period.class));
130         addSerializer(Year.class, YearSerializer.INSTANT);
131         addSerializer(YearMonth.class, YearMonthSerializer.INSTANT);
132
133         /* 27-Jun-2015, tatu: This is the real difference from the old
134          * @link JSR310Module): default is to produce ISO-8601 compatible
135          */
136     }
137 }
```

```

J: JavaTimeModule.java
81 public final class BQTimeModule extends SimpleModule {
105 }
106
107 @Override
108 public void setupModule(SetupContext context) {
109     super.setupModule(context);
110     context.addValueInstantiators(new ValueInstantiators.Base() {
111         @Override
112         public ValueInstantiator findValueInstantiator(DeserializationConfig config,
113             BeanDescription beanDesc, ValueInstantiator defaultInstantiator) {
114             JavaType type = beanDesc.getType();
115             Class<?> raw = type.getRawClass();
116
117             // 15-May-2015, tatu: In theory not safe, but in practice we do need to do
118             // because we will (for now) be getting a subtype, but in future may want to
119             // to the common base type. Even more, serializer may purposefully force us
120             // So... in practice it really should always work, in the end. :)
121             if (ZoneId.class.isAssignableFrom(raw)) {
122                 // let's assume we should be getting "empty" StdValueInstantiator here:
123                 if (defaultInstantiator instanceof StdValueInstantiator) {
124                     StdValueInstantiator inst = (StdValueInstantiator) defaultInstantiator;
125                     // one further complication: we need ZoneId info, not sub-class
126                     AnnotatedClass ac;
127                     if (raw == ZoneId.class) {
128                         ac = beanDesc.getClassInfo();
129                     } else {
130                         // we don't need Annotations, so constructing directly is fine
131                         // even if it's not generally recommended
132                         ac = AnnotatedClassResolver.resolve(config,
133                             config.constructType(ZoneId.class), config);
134                     }
135                     if (!inst.canCreateFromString()) {
136                         AnnotatedMethod factory = _findFactory(ac, "of", String.class);
137                         if (factory != null) {
138                             inst.configureFromStringCreator(factory);
139                         }
140                     }
141                     // otherwise... should we indicate an error?
142                     // return ZoneIdInstantiator.construct(config, beanDesc, defaultInstantiator);
143                 }
144             }
145             return defaultInstantiator;
146         }
147     });
148 }
149
150- protected AnnotatedMethod _findFactory(AnnotatedClass cls, String name, Class<?>... args) {
151     final int argCount = args.length;
152     for (AnnotatedMethod method : cls.getFactoryMethods()) {
153         if (!name.equals(method.getName())) {
154             continue;
155         }
156         if ((method.getParameterCount() != argCount)) {
157             continue;
158         }
159         for (int i = 0; i < argCount; ++i) {
160             Class<?> argType = method.getParameter(i).getRawType();
161             if (!argType.isAssignableFrom(argsTypes[i])) {
162                 continue;
163             }
164         }
165         return method;
166     }
167     return null;
168 }

```

```

129 {
130 {
131     /* 27-Jun-2015, tatu: This is the real difference from the old
132     * But this is configurable.
133     */
134     addSerializer(ZoneDateTime.class, ZoneDateTimeSerializer.INSTANCE);
135     // note: actual concrete type is 'ZoneRegion', but that's not visible:
136     addSerializer(ZoneId.class, new ToStringSerializer(ZoneId.class));
137     addSerializer(ZoneOffset.class, new ToStringSerializer(ZoneOffset.class));
138     // key serializers
139     addKeySerializer(ZoneDateTime.class, ZoneDateTimeKeySerializer.INSTANCE);
140     // key deserializers
141     addKeyDeserializer(Duration.class, DurationKeyDeserializer.INSTANCE);
142     addKeyDeserializer(Instant.class, InstantKeyDeserializer.INSTANCE);
143     addKeyDeserializer(LocalDate.class, LocalDateKeyDeserializer.INSTANCE);
144     addKeyDeserializer(LocalDate.class, LocalDateKeyDeserializer.INSTANCE);
145     addKeyDeserializer(LocalTime.class, LocalTimeKeyDeserializer.INSTANCE);
146     addKeyDeserializer(MonthDay.class, MonthDayKeyDeserializer.INSTANCE);
147     addKeyDeserializer(OffsetDateTime.class, OffsetDateTimeKeyDeserializer.INSTANCE);
148     addKeyDeserializer(OffsetTime.class, OffsetTimeKeyDeserializer.INSTANCE);
149     addKeyDeserializer(Period.class, PeriodKeyDeserializer.INSTANCE);
150     addKeyDeserializer(Year.class, YearKeyDeserializer.INSTANCE);
151     addKeyDeserializer(YearMonth.class, YearMonthKeyDeserializer.INSTANCE);
152     addKeyDeserializer(ZoneDateTime.class, ZoneDateTimeKeyDeserializer.INSTANCE);
153     addKeyDeserializer(ZoneId.class, ZoneIdKeyDeserializer.INSTANCE);
154     addKeyDeserializer(ZoneOffset.class, ZoneOffsetKeyDeserializer.INSTANCE);
155 }
156
157 @Override
158 public void setupModule(SetupContext context) {
159     super.setupModule(context);
160     context.addValueInstantiators(new ValueInstantiators.Base() {
161         @Override
162         public ValueInstantiator findValueInstantiator(DeserializationConfig config,
163             BeanDescription beanDesc, ValueInstantiator defaultInstantiator) {
164             JavaType type = beanDesc.getType();
165             Class<?> raw = type.getRawClass();
166
167             // 15-May-2015, tatu: In theory not safe, but in practice we do need to do "fuzz"
168             // because we will (for now) be getting a subtype, but in future may want to do
169             // to the common base type. Even more, serializer may purposefully force use of
170             // So... in practice it really should always work, in the end. :)
171             if (ZoneId.class.isAssignableFrom(raw)) {
172                 // let's assume we should be getting "empty" StdValueInstantiator here:
173                 if (defaultInstantiator instanceof StdValueInstantiator) {
174                     StdValueInstantiator inst = (StdValueInstantiator) defaultInstantiator;
175                     // one further complication: we need ZoneId info, not sub-class
176                     AnnotatedClass ac;
177                     if (raw == ZoneId.class) {
178                         ac = beanDesc.getClassInfo();
179                     } else {
180                         // we don't need Annotations, so constructing directly is fine here
181                         // even if it's not generally recommended
182                         ac = AnnotatedClassResolver.resolve(config,
183                             config.constructType(ZoneId.class), config);
184                     }
185                     if (!inst.canCreateFromString()) {
186                         AnnotatedMethod factory = _findFactory(ac, "of", String.class);
187                         if (factory != null) {
188                             inst.configureFromStringCreator(factory);
189                         }
190                     }
191                     // otherwise... should we indicate an error?
192                     // return ZoneIdInstantiator.construct(config, beanDesc, defaultInstantiator);
193                 }
194             }
195             return defaultInstantiator;
196         }
197     });
198 }
199
200- protected AnnotatedMethod _findFactory(AnnotatedClass cls, String name, Class<?>... args) {
201     final int argCount = args.length;
202     for (AnnotatedMethod method : cls.getFactoryMethods()) {
203         if (!name.equals(method.getName())) {
204             continue;
205         }
206         if ((method.getParameterCount() != argCount)) {
207             continue;
208         }
209         for (int i = 0; i < argCount; ++i) {
210             Class<?> argType = method.getParameter(i).getRawType();
211             if (!argType.isAssignableFrom(argsTypes[i])) {
212                 continue;
213             }
214         }
215         return method;
216     }
217     return null;
218 }

```

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.



```

J DB2Driver.java
1  /*
2  * Copyright (C) 2020 IBM Corporation
3  *
4  * Licensed under the Apache License, Version 2.0 (the "License");
5  * you may not use this file except in compliance with the License.
6  * You may obtain a copy of the License at
7  *
8  * http://www.apache.org/licenses/LICENSE-2.0
9  *
10 * Unless required by applicable law or agreed to in writing, software
11 * distributed under the License is distributed on an "AS IS" BASIS,
12 * WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
13 * See the License for the specific language governing permissions and
14 * limitations under the License.
15 */
16 package io.vertx.mysqlclient.spi;
17
18 import io.vertx.core.Future;
19 import io.vertx.core.Handler;
20 import io.vertx.core.Vertx;
21 import io.vertx.core.impl.CloseFuture;
22 import io.vertx.core.impl.ContextInternal;
23 import io.vertx.core.impl.VertxInternal;
24 import io.vertx.core.json.JsonObject;
25 import io.vertx.core.net.NetClientOptions;
26 import io.vertx.mysqlclient.MySQLConnectOptions;
27 import io.vertx.mysqlclient.impl.*;
28 import io.vertx.sqlclient.Pool;
29 import io.vertx.sqlclient.PoolOptions;
30 import io.vertx.sqlclient.SqlConnectOptions;
31 import io.vertx.sqlclient.SqlConnection;
32 import io.vertx.sqlclient.impl.Connection;
33 import io.vertx.sqlclient.impl.CloseablePool;
34 import io.vertx.sqlclient.impl.PoolImpl;
35 import io.vertx.sqlclient.impl.SqlConnectionInternal;
36 import io.vertx.sqlclient.spi.ConnectionFactory;
37 import io.vertx.sqlclient.spi.Driver;
38
39 import java.util.function.Supplier;
40
41 public class MySQLDriver implements Driver<MySQLConnectOptions> {
42
43   private static final String SHARED_CLIENT_KEY = "_vertx.shared.mysqlclient";
44
45   public static final MySQLDriver INSTANCE = new MySQLDriver();
46
47   @Override
48   public MySQLConnectOptions downcast(SqlConnectOptions connectOptions) {
49     return connectOptions instanceof MySQLConnectOptions ? (MySQLConnectOptions) connectOptions : null;
50   }
51
52   @Override
53   public Pool newPool(Vertx vertx, Supplier<Future<MySQLConnectOptions>> databases, PoolOptions poolOptions) {
54     VertxInternal vx = (VertxInternal) vertx;
55     PoolImpl pool;
56     if (options.isShared()) {
57       pool = vx.createSharedResource(SHARED_CLIENT_KEY, options.getName(), closeFuture, cf -> new MySQLConnectOptions(options));
58     } else {
59       pool = new PoolImpl(vx, connectHandler, databases, options, transportOptions, closeFuture);
60     }
61     return new CloseablePool<>(vx, closeFuture, pool);
62   }
63
64   private PoolImpl newPoolImpl(VertxInternal vertx, Handler<SqlConnection> connectHandler,
65                               boolean pipelinedPool = poolOptions instanceof MySQLPoolOptions && ((MySQLPoolOptions) poolOptions).isPipelined(),
66                               ConnectionFactory<MySQLConnectOptions> factory = createConnectionFactory(vertx, transportOptions),
67                               int pipeliningLimit = pipelinedPool ? baseConnectOptions.getPipeliningLimit() : 1,
68                               PoolImpl pool = new PoolImpl(vertx, this, tracer, metrics, pipeliningLimit, options, null, factory),
69                               List<ConnectionFactory> lst = databases.stream().map(o -> createConnectionFactory(vertx, o))::toList,
70                               ConnectionFactory factory = ConnectionFactory.roundRobinSelector(lst);
71   pool.connectionProvider(factory::connect);
72   pool.init();
73   closeFuture.add(factory);
74   return pool;
75 }
76
77 @Override
78 public MySQLConnectOptions parseConnectionUri(String uri) {
79   JsonObject conf = MySQLConnectionUriParser.parse(uri, false);
80   return conf == null ? null : new MySQLConnectOptions(conf);
81 }
82
83 @Override
84 public boolean acceptsOptions(SqlConnectOptions options) {
85   return options instanceof MySQLConnectOptions || SqlConnectOptions.class.equals(options.getClass());
86 }
87
88 @Override
89 public ConnectionFactory<MySQLConnectOptions> createConnectionFactory(Vertx vertx, NetClientOptions netClientOptions) {
90   return new MySQLConnectionFactory((VertxInternal) vertx);
91 }
92
93 @Override
94 public SqlConnectionInternal wrapConnection(ContextInternal context, ConnectionFactory<MySQLConnectOptions> factory, com.mysql.cj.jdbc.Driver com) {
95   return new MySQLConnectionImpl(context, factory, com);
96 }
97 }
98
99 J DB2Driver.java
100 /*
101 * Copyright (C) 2020 IBM Corporation
102 *
103 * Licensed under the Apache License, Version 2.0 (the "License");
104 * you may not use this file except in compliance with the License.
105 * You may obtain a copy of the License at
106 *
107 * http://www.apache.org/licenses/LICENSE-2.0
108 *
109 * Unless required by applicable law or agreed to in writing, software
110 * distributed under the License is distributed on an "AS IS" BASIS,
111 * WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
112 * See the License for the specific language governing permissions and
113 * limitations under the License.
114 */
115 package io.vertx.db2client.spi;
116
117 import io.vertx.core.Vertx;
118 import io.vertx.core.impl.CloseFuture;
119 import io.vertx.core.impl.ContextInternal;
120 import io.vertx.core.impl.VertxInternal;
121 import io.vertx.core.json.JsonObject;
122 import io.vertx.core.metrics.ClientMetrics;
123 import io.vertx.core.metrics.MetricNames;
124 import io.vertx.db2client.DB2ConnectOptions;
125 import io.vertx.db2client.DB2Pool;
126 import io.vertx.db2client.impl.*;
127 import io.vertx.sqlclient.PoolOptions;
128 import io.vertx.sqlclient.SqlConnectOptions;
129 import io.vertx.sqlclient.impl.Connection;
130 import io.vertx.sqlclient.impl.PoolImpl;
131 import io.vertx.sqlclient.impl.SqlConnectionInternal;
132 import io.vertx.sqlclient.impl.tracing.QueryTracer;
133 import io.vertx.sqlclient.spi.ConnectionFactory;
134 import io.vertx.sqlclient.spi.Driver;
135
136 import java.util.List;
137 import java.util.stream.Collectors;
138
139 public class DB2Driver implements Driver {
140
141   private static final String SHARED_CLIENT_KEY = "_vertx.shared.db2client";
142
143   public static final DB2Driver INSTANCE = new DB2Driver();
144
145   @Override
146   public DB2Pool newPool(Vertx vertx, List<? extends SqlConnectOptions> databases, PoolOptions poolOptions) {
147     VertxInternal vx = (VertxInternal) vertx;
148     PoolImpl pool;
149     if (options.isShared()) {
150       pool = vx.createSharedClient(SHARED_CLIENT_KEY, options.getName(), closeFuture, cf -> new DB2ConnectOptions(options));
151     } else {
152       pool = new PoolImpl(vx, databases, options, closeFuture);
153     }
154     return new DB2PoolImpl(vx, closeFuture, pool);
155   }
156
157   private PoolImpl newPoolImpl(VertxInternal vertx, List<? extends SqlConnectOptions> databases,
158                               DB2ConnectOptions baseConnectOptions = DB2ConnectOptions.wrap(databases.get(0)),
159                               QueryTracer tracer = vertx.tracer() == null ? null : new QueryTracer(vertx.tracer(), baseConnectOptions),
160                               VertxMetrics vertxMetrics = vertx.metricsSPI(),
161                               ClientMetrics metrics = vertxMetrics != null ? vertxMetrics.createClientMetrics(baseConnectOptions) : null,
162                               boolean pipelinedPool = options instanceof DB2PoolOptions && ((DB2PoolOptions) options).isPipelined(),
163                               int pipeliningLimit = pipelinedPool ? baseConnectOptions.getPipeliningLimit() : 1,
164                               PoolImpl pool = new PoolImpl(vertx, this, tracer, metrics, pipeliningLimit, options, null, factory),
165                               List<ConnectionFactory> lst = databases.stream().map(o -> createConnectionFactory(vertx, o))::toList,
166                               ConnectionFactory factory = ConnectionFactory.roundRobinSelector(lst);
167   pool.connectionProvider(factory::connect);
168   pool.init();
169   closeFuture.add(factory);
170   return pool;
171 }
172
173 @Override
174 public DB2ConnectOptions parseConnectionUri(String uri) {
175   JsonObject conf = DB2ConnectionUriParser.parse(uri, false);
176   return conf == null ? null : new DB2ConnectOptions(conf);
177 }
178
179 @Override
180 public boolean acceptsOptions(SqlConnectOptions options) {
181   return options instanceof DB2ConnectOptions || SqlConnectOptions.class.equals(options.getClass());
182 }
183
184 @Override
185 public ConnectionFactory createConnectionFactory(Vertx vertx, SqlConnectOptions database) {
186   return new DB2ConnectionFactory((VertxInternal) vertx, DB2ConnectOptions.wrap(database));
187 }
188
189 @Override
190 public SqlConnectionInternal wrapConnection(ContextInternal context, ConnectionFactory factory, com.ibm.db2.jcc.DB2Connection conn, tracer, metrics) {
191   return new DB2ConnectionImpl(context, factory, conn, tracer, metrics);
192 }
193 }

```

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/PrimitiveColumnTypeI
nfo.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.

```

J BasicTypeInfo.java
1  /*
2  * Copyright 2022 Bytedance Ltd. and/or its affiliates.
3  *
4  * Licensed under the Apache License, Version 2.0 (the "License");
5  * you may not use this file except in compliance with the License.
6  * You may obtain a copy of the License at
7  *
8  * http://www.apache.org/licenses/LICENSE-2.0
9  *
10 * Unless required by applicable law or agreed to in writing, software
11 * distributed under the License is distributed on an "AS IS" BASIS,
12 * WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
13 * See the License for the specific language governing permissions and
14 * limitations under the License.
15 */
16
17 package com.bytedance.bitsail.flink.core.typeinfo;
18
19 import com.bytedance.bitsail.common.column.BooleanColumn;
20 import com.bytedance.bitsail.common.column.BytesColumn;
21 import com.bytedance.bitsail.common.column.DateColumn;
22 import com.bytedance.bitsail.common.column.DoubleColumn;
23 import com.bytedance.bitsail.common.column.LongColumn;
24 import com.bytedance.bitsail.common.column.StringColumn;
25 import com.bytedance.bitsail.flink.core.typeutils.base.BoolColumnComparator;
26 import com.bytedance.bitsail.flink.core.typeutils.base.BoolColumnSerializer;
27 import com.bytedance.bitsail.flink.core.typeutils.base.BytesColumnComparator;
28 import com.bytedance.bitsail.flink.core.typeutils.base.BytesColumnSerializer;
29 import com.bytedance.bitsail.flink.core.typeutils.base.DateColumnComparator;
30 import com.bytedance.bitsail.flink.core.typeutils.base.DateColumnSerializer;
31 import com.bytedance.bitsail.flink.core.typeutils.base.DoubleColumnComparator;
32 import com.bytedance.bitsail.flink.core.typeutils.base.DoubleColumnSerializer;
33 import com.bytedance.bitsail.flink.core.typeutils.base.LongColumnComparator;
34 import com.bytedance.bitsail.flink.core.typeutils.base.LongColumnSerializer;
35 import com.bytedance.bitsail.flink.core.typeutils.base.StringColumnComparator;
36 import com.bytedance.bitsail.flink.core.typeutils.base.StringColumnSerializer;
37
38 import org.apache.flink.annotation.PublicEvolving;
39 import org.apache.flink.api.common.ExecutionConfig;
40 import org.apache.flink.api.common.functions.InvalidTypesException;
41 import org.apache.flink.api.common.typeinfo.AtomicType;
42 import org.apache.flink.api.common.typeinfo.BasicTypeInfo;
43 import org.apache.flink.api.common.typeinfo.TypeInformation;
44 import org.apache.flink.api.common.typeutils.TypeComparator;
45 import org.apache.flink.api.common.typeutils.TypeSerializer;
46
47 import java.lang.reflect.Constructor;
48 import java.util.Arrays;
49
50 import java.util.HashMap;
51 import java.util.Map;
52 import java.util.Objects;
53
54

```

```

1  /*
2  * Licensed to the Apache Software Foundation (ASF) under one
3  * or more contributor license agreements. See the NOTICE file
4  * distributed with this work for additional information
5  * regarding copyright ownership. The ASF licenses this file
6  * to you under the Apache License, Version 2.0 (the
7  * "License"); you may not use this file except in compliance
8  * with the License. You may obtain a copy of the License at
9  *
10 * http://www.apache.org/licenses/LICENSE-2.0
11 *
12 * Unless required by applicable law or agreed to in writing, software
13 * distributed under the License is distributed on an "AS IS" BASIS,
14 * WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
15 * See the License for the specific language governing permissions and
16 * limitations under the License.
17 */
18
19 package org.apache.flink.api.common.typeinfo;
20
21
22
23
24
25
26
27

```



J BasicTypeInfo.java

```
53- import static org.apache.flink.util.Preconditions.checkNotNull;
54-
55- /**
56-  * @desc:
57-  */
58- @Deprecated
59-
60- public class PrimitiveColumnTypeInfo<T> extends TypeInformation<T> implements AtomicType<T> {
61-     public static final PrimitiveColumnTypeInfo<StringColumn> STRING_COLUMN_TYPE_INFO = new PrimitiveColumnTypeInfo<StringColumn>(
62-         StringColumnSerializer.INSTANCE, StringColumnComparator.class);
63-     public static final PrimitiveColumnTypeInfo<BooleanColumn> BOOL_COLUMN_TYPE_INFO = new PrimitiveColumnTypeInfo<BooleanColumn>(
64-         BooleanColumnSerializer.INSTANCE, BooleanColumnComparator.class);
65-     public static final PrimitiveColumnTypeInfo<BytesColumn> BYTES_COLUMN_TYPE_INFO = new PrimitiveColumnTypeInfo<BytesColumn>(
66-         BytesColumnSerializer.INSTANCE, BytesColumnComparator.class);
67-     public static final PrimitiveColumnTypeInfo<LongColumn> LONG_COLUMN_TYPE_INFO = new PrimitiveColumnTypeInfo<LongColumn>(
68-         LongColumnSerializer.INSTANCE, LongColumnComparator.class);
69-     public static final PrimitiveColumnTypeInfo<DateColumn> DATE_COLUMN_TYPE_INFO = new PrimitiveColumnTypeInfo<DateColumn>(
70-         DateColumnSerializer.INSTANCE, DateColumnComparator.class);
71-     public static final PrimitiveColumnTypeInfo<DoubleColumn> DOUBLE_COLUMN_TYPE_INFO = new PrimitiveColumnTypeInfo<DoubleColumn>(
72-         DoubleColumnSerializer.INSTANCE, DoubleColumnComparator.class);
73-     private static final long serialVersionUID = 1L;
74-     private static final Map<Class<T>, PrimitiveColumnTypeInfo<T>> TYPES = new HashMap<>();
75-
76-     static {
77-         TYPES.put(StringColumn.class, STRING_COLUMN_TYPE_INFO);
78-         TYPES.put(BooleanColumn.class, BOOL_COLUMN_TYPE_INFO);
79-         TYPES.put(LongColumn.class, LONG_COLUMN_TYPE_INFO);
80-         TYPES.put(DoubleColumn.class, DOUBLE_COLUMN_TYPE_INFO);
81-         TYPES.put(DateColumn.class, DATE_COLUMN_TYPE_INFO);
82-     }
83-
84-     private final Class<T> clazz;
85-     private final TypeSerializer<T> serializer;
86-     private final Class<T>[] possibleCastTargetTypes;
87-     private final Class<T> extends TypeComparator<T> comparatorClass;
88-
89-     protected PrimitiveColumnTypeInfo(Class<T> clazz, Class<T>[] possibleCastTargetTypes, TypeSerializer<T> serializer,
90-         Class<T> extends TypeComparator<T> comparatorClass) {
91-         this.clazz = Preconditions.checkNotNull(clazz);
92-         this.possibleCastTargetTypes = Preconditions.checkNotNull(possibleCastTargetTypes);
93-         this.serializer = Preconditions.checkNotNull(serializer);
94-         // comparator can be null as in VOID_TYPE_INFO
95-         this.comparatorClass = comparatorClass;
96-
97-         private static <K> TypeComparator<K> instantiateComparator(Class<T> extends TypeComparator<T> comparatorClass) {
98-             try {
99-                 Constructor<T> extends TypeComparator<K> constructor = comparatorClass.getConstructor();
100-                 return constructor.newInstance(ascendingOrder);
101-             } catch (Exception e) {
102-                 throw new RuntimeException("Could not initialize basic comparator " + comparatorClass);
103-             }
104-         }
105-
106-         @PublicEvolving
107-         public static <K> PrimitiveColumnTypeInfo<K> getFor(Class<K> type) {
108-             if (type == null) {
109-                 throw new NullPointerException();
110-             }
111-             @SuppressWarnings("unchecked")
112-             PrimitiveColumnTypeInfo<K> info = (PrimitiveColumnTypeInfo<K>) TYPES.get(type);
113-             return info;
114-         }
115-
116-         /**
117-          * Returns whether this type should be automatically casted to
118-          * the target type in an arithmetic operation.
119-          */
120-         @PublicEvolving
121-         public boolean shouldAutoCastTo(BasicTypeInfo<T> to) {
122-             for (Class<T> possibleTo : possibleCastTargetTypes) {
123-                 if (possibleTo.equals(to.getTypeClass())) {
124-                     return true;
125-                 }
126-             }
127-             return false;
128-         }
129-
130-         @Override
131-         public boolean equals(Object o) {
132-             if (o == null) {
133-                 return false;
134-             }
135-             if (o instanceof PrimitiveColumnTypeInfo) {
136-                 PrimitiveColumnTypeInfo<T> other = (PrimitiveColumnTypeInfo<T>) o;
137-                 return this.clazz.equals(other.clazz) && this.serializer.equals(other.serializer) && this.comparatorClass.equals(other.comparatorClass);
138-             }
139-             return false;
140-         }
141-
142-         @Override
143-         public int hashCode() {
144-             return Objects.hash(clazz, serializer, comparatorClass);
145-         }
146-     }
147- }
```

J BasicTypeInfo.java

```
59- public class PrimitiveColumnTypeInfo<T> extends TypeInformation<T> implements AtomicType<T> {
60-     private final Class<T> clazz;
61-     private final TypeSerializer<T> serializer;
62-     private final Class<T>[] possibleCastTargetTypes;
63-     private final Class<T> extends TypeComparator<T> comparatorClass;
64-
65-     protected PrimitiveColumnTypeInfo(Class<T> clazz, Class<T>[] possibleCastTargetTypes, TypeSerializer<T> serializer,
66-         Class<T> extends TypeComparator<T> comparatorClass) {
67-         this.clazz = Preconditions.checkNotNull(clazz);
68-         this.possibleCastTargetTypes = Preconditions.checkNotNull(possibleCastTargetTypes);
69-         this.serializer = Preconditions.checkNotNull(serializer);
70-         // comparator can be null as in VOID_TYPE_INFO
71-         this.comparatorClass = comparatorClass;
72-
73-         private static <K> TypeComparator<K> instantiateComparator(Class<T> extends TypeComparator<T> comparatorClass) {
74-             try {
75-                 Constructor<T> extends TypeComparator<K> constructor = comparatorClass.getConstructor();
76-                 return constructor.newInstance(ascendingOrder);
77-             } catch (Exception e) {
78-                 throw new RuntimeException("Could not initialize basic comparator " + comparatorClass);
79-             }
80-         }
81-
82-         @PublicEvolving
83-         public static <K> PrimitiveColumnTypeInfo<K> getFor(Class<K> type) {
84-             if (type == null) {
85-                 throw new NullPointerException();
86-             }
87-             @SuppressWarnings("unchecked")
88-             PrimitiveColumnTypeInfo<K> info = (PrimitiveColumnTypeInfo<K>) TYPES.get(type);
89-             return info;
90-         }
91-
92-         /**
93-          * Returns whether this type should be automatically casted to
94-          * the target type in an arithmetic operation.
95-          */
96-         @PublicEvolving
97-         public boolean shouldAutoCastTo(BasicTypeInfo<T> to) {
98-             for (Class<T> possibleTo : possibleCastTargetTypes) {
99-                 if (possibleTo.equals(to.getTypeClass())) {
100-                     return true;
101-                 }
102-             }
103-             return false;
104-         }
105-
106-         @Override
107-         public boolean equals(Object o) {
108-             if (o == null) {
109-                 return false;
110-             }
111-             if (o instanceof PrimitiveColumnTypeInfo) {
112-                 PrimitiveColumnTypeInfo<T> other = (PrimitiveColumnTypeInfo<T>) o;
113-                 return this.clazz.equals(other.clazz) && this.serializer.equals(other.serializer) && this.comparatorClass.equals(other.comparatorClass);
114-             }
115-             return false;
116-         }
117-
118-         @Override
119-         public int hashCode() {
120-             return Objects.hash(clazz, serializer, comparatorClass);
121-         }
122-     }
123- }
```

```
28- import com.google.common.base.Preconditions;
29- import org.apache.flink.api.common.ExecutionConfig;
30- import org.apache.flink.api.common.functions.InvalidTypesException;
31- import org.apache.flink.api.common.typeutils.TypeComparator;
32- import org.apache.flink.api.common.typeutils.TypeSerializer;
33- import org.apache.flink.api.common.typeutils.base.BooleanComparator;
34- import org.apache.flink.api.common.typeutils.base.BooleanSerializer;
35- import org.apache.flink.api.common.typeutils.base.ByteComparator;
36- import org.apache.flink.api.common.typeutils.base.ByteSerializer;
37- import org.apache.flink.api.common.typeutils.base.CharComparator;
38- import org.apache.flink.api.common.typeutils.base.CharSerializer;
39- import org.apache.flink.api.common.typeutils.base.DateComparator;
40- import org.apache.flink.api.common.typeutils.base.DateSerializer;
41- import org.apache.flink.api.common.typeutils.base.DoubleComparator;
42- import org.apache.flink.api.common.typeutils.base.DoubleSerializer;
43- import org.apache.flink.api.common.typeutils.base.FloatComparator;
44- import org.apache.flink.api.common.typeutils.base.FloatSerializer;
45- import org.apache.flink.api.common.typeutils.base.IntComparator;
46- import org.apache.flink.api.common.typeutils.base.IntSerializer;
47- import org.apache.flink.api.common.typeutils.base.LongComparator;
48- import org.apache.flink.api.common.typeutils.base.LongSerializer;
49- import org.apache.flink.api.common.typeutils.base.ShortComparator;
50- import org.apache.flink.api.common.typeutils.base.ShortSerializer;
51- import org.apache.flink.api.common.typeutils.base.StringComparator;
52- import org.apache.flink.api.common.typeutils.base.StringSerializer;
53- import org.apache.flink.api.common.typeutils.base.VoidSerializer;
54-
55- /**
56-  * Type information for primitive types (int, long, double, byte, ...), String, Date, and Void.
57-  */
58- public class BasicTypeInfo<T> extends TypeInformation<T> implements AtomicType<T> {
59-     private static final long serialVersionUID = -438955228409131770L;
60-
61-     public static final BasicTypeInfo<String> STRING_TYPE_INFO = new BasicTypeInfo<String>(String.class,
62-         StringSerializer.INSTANCE, StringComparator.class);
63-     public static final BasicTypeInfo<Boolean> BOOLEAN_TYPE_INFO = new BasicTypeInfo<Boolean>(Boolean.class,
64-         BooleanSerializer.INSTANCE, BooleanComparator.class);
65-     public static final BasicTypeInfo<Byte> BYTE_TYPE_INFO = new IntegerTypeInfo<Byte>(Byte.class,
66-         IntegerSerializer.INSTANCE, IntegerComparator.class);
67-     public static final BasicTypeInfo<Short> SHORT_TYPE_INFO = new IntegerTypeInfo<Short>(Short.class,
68-         IntegerSerializer.INSTANCE, IntegerComparator.class);
69-     public static final BasicTypeInfo<Integer> INT_TYPE_INFO = new IntegerTypeInfo<Integer>(Integer.class,
70-         IntegerSerializer.INSTANCE, IntegerComparator.class);
71-     public static final BasicTypeInfo<Long> LONG_TYPE_INFO = new IntegerTypeInfo<Long>(Long.class,
72-         IntegerSerializer.INSTANCE, IntegerComparator.class);
73-     public static final BasicTypeInfo<Float> FLOAT_TYPE_INFO = new FractionalTypeInfo<Float>(Float.class,
74-         FractionalSerializer.INSTANCE, FractionalComparator.class);
75-     public static final BasicTypeInfo<Double> DOUBLE_TYPE_INFO = new FractionalTypeInfo<Double>(Double.class,
76-         FractionalSerializer.INSTANCE, FractionalComparator.class);
77-     public static final BasicTypeInfo<Character> CHAR_TYPE_INFO = new BasicTypeInfo<Character>(Character.class,
78-         CharacterSerializer.INSTANCE, CharacterComparator.class);
79-     public static final BasicTypeInfo<Date> DATE_TYPE_INFO = new BasicTypeInfo<Date>(Date.class,
80-         DateSerializer.INSTANCE, DateComparator.class);
81-     public static final BasicTypeInfo<Void> VOID_TYPE_INFO = new BasicTypeInfo<Void>(Void.class,
82-         VoidSerializer.INSTANCE, VoidComparator.class);
83-
84-     private final Class<T> clazz;
85-     private final TypeSerializer<T> serializer;
86-     private final Class<T>[] possibleCastTargetTypes;
87-     private final Class<T> extends TypeComparator<T> comparatorClass;
88-
89-     protected BasicTypeInfo(Class<T> clazz, Class<T>[] possibleCastTargetTypes, TypeSerializer<T> serializer,
90-         Class<T> extends TypeComparator<T> comparatorClass) {
91-         this.clazz = Preconditions.checkNotNull(clazz);
92-         this.possibleCastTargetTypes = Preconditions.checkNotNull(possibleCastTargetTypes);
93-         this.serializer = Preconditions.checkNotNull(serializer);
94-         // comparator can be null as in VOID_TYPE_INFO
95-         this.comparatorClass = comparatorClass;
96-
97-         private static <K> TypeComparator<K> instantiateComparator(Class<T> extends TypeComparator<T> comparatorClass) {
98-             try {
99-                 Constructor<T> extends TypeComparator<K> constructor = comparatorClass.getConstructor();
100-                 return constructor.newInstance(ascendingOrder);
101-             } catch (Exception e) {
102-                 throw new RuntimeException("Could not initialize basic comparator " + comparatorClass);
103-             }
104-         }
105-
106-         @PublicEvolving
107-         public static <K> BasicTypeInfo<K> getFor(Class<K> type) {
108-             if (type == null) {
109-                 throw new NullPointerException();
110-             }
111-             @SuppressWarnings("unchecked")
112-             BasicTypeInfo<K> info = (BasicTypeInfo<K>) TYPES.get(type);
113-             return info;
114-         }
115-
116-         /**
117-          * Returns whether this type should be automatically casted to
118-          * the target type in an arithmetic operation.
119-          */
120-         @PublicEvolving
121-         public boolean shouldAutoCastTo(BasicTypeInfo<T> to) {
122-             for (Class<T> possibleTo : possibleCastTargetTypes) {
123-                 if (possibleTo.equals(to.getTypeClass())) {
124-                     return true;
125-                 }
126-             }
127-             return false;
128-         }
129-
130-         @Override
131-         public boolean equals(Object o) {
132-             if (o == null) {
133-                 return false;
134-             }
135-             if (o instanceof BasicTypeInfo) {
136-                 BasicTypeInfo<T> other = (BasicTypeInfo<T>) o;
137-                 return this.clazz.equals(other.clazz) && this.serializer.equals(other.serializer) && this.comparatorClass.equals(other.comparatorClass);
138-             }
139-             return false;
140-         }
141-
142-         @Override
143-         public int hashCode() {
144-             return Objects.hash(clazz, serializer, comparatorClass);
145-         }
146-     }
147- }
```

```
59- public class BasicTypeInfo<T> extends TypeInformation<T> implements AtomicType<T> {
60-     private final Class<T> clazz;
61-     private final TypeSerializer<T> serializer;
62-     private final Class<T>[] possibleCastTargetTypes;
63-     private final Class<T> extends TypeComparator<T> comparatorClass;
64-
65-     protected BasicTypeInfo(Class<T> clazz, Class<T>[] possibleCastTargetTypes, TypeSerializer<T> serializer,
66-         Class<T> extends TypeComparator<T> comparatorClass) {
67-         this.clazz = Preconditions.checkNotNull(clazz);
68-         this.possibleCastTargetTypes = Preconditions.checkNotNull(possibleCastTargetTypes);
69-         this.serializer = Preconditions.checkNotNull(serializer);
70-         // comparator can be null as in VOID_TYPE_INFO
71-         this.comparatorClass = comparatorClass;
72-
73-         private static <K> TypeComparator<K> instantiateComparator(Class<T> extends TypeComparator<T> comparatorClass) {
74-             try {
75-                 Constructor<T> extends TypeComparator<K> constructor = comparatorClass.getConstructor();
76-                 return constructor.newInstance(ascendingOrder);
77-             } catch (Exception e) {
78-                 throw new RuntimeException("Could not initialize basic comparator " + comparatorClass);
79-             }
80-         }
81-
82-         @PublicEvolving
83-         public static <K> BasicTypeInfo<K> getFor(Class<K> type) {
84-             if (type == null) {
85-                 throw new NullPointerException();
86-             }
87-             @SuppressWarnings("unchecked")
88-             BasicTypeInfo<K> info = (BasicTypeInfo<K>) TYPES.get(type);
89-             return info;
90-         }
91-
92-         /**
93-          * Returns whether this type should be automatically casted to
94-          * the target type in an arithmetic operation.
95-          */
96-         @PublicEvolving
97-         public boolean shouldAutoCastTo(BasicTypeInfo<T> to) {
98-             for (Class<T> possibleTo : possibleCastTargetTypes) {
99-                 if (possibleTo.equals(to.getTypeClass())) {
100-                     return true;
101-                 }
102-             }
103-             return false;
104-         }
105-
106-         @Override
107-         public boolean equals(Object o) {
108-             if (o == null) {
109-                 return false;
110-             }
111-             if (o instanceof BasicTypeInfo) {
112-                 BasicTypeInfo<T> other = (BasicTypeInfo<T>) o;
113-                 return this.clazz.equals(other.clazz) && this.serializer.equals(other.serializer) && this.comparatorClass.equals(other.comparatorClass);
114-             }
115-             return false;
116-         }
117-
118-         @Override
119-         public int hashCode() {
120-             return Objects.hash(clazz, serializer, comparatorClass);
121-         }
122-     }
123- }
```

J BasicTypeInfo.java

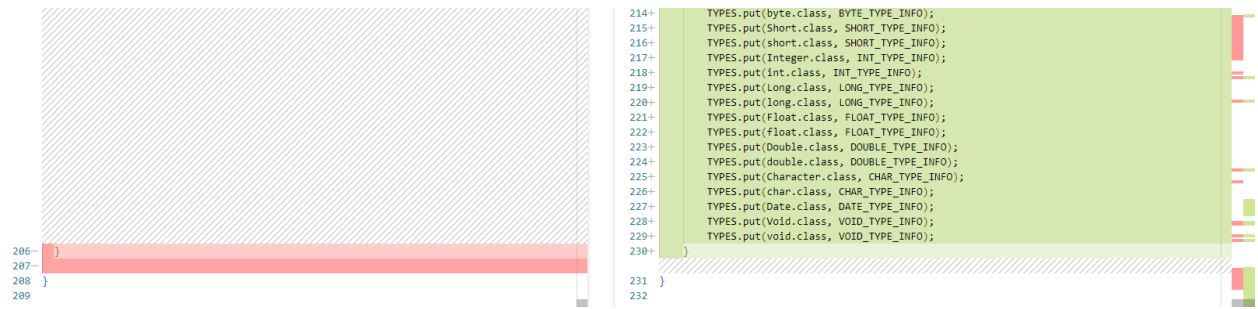
```
59 public class PrimitiveColumnInfo<T> extends TypeInformation<T> implements AtomicType<T> {
133 @Override
134 public boolean isBasicType() {
135     return false;
136 }
137
138 @Override
139 public boolean isTupleType() {
140     return false;
141 }
142
143 @Override
144 public int getArity() {
145     return 1;
146 }
147
148 @Override
149 public int getTotalFields() {
150     return 1;
151 }
152
153 @Override
154 public Class<T> getTypeClass() {
155     return this.clazz;
156 }
157
158 @Override
159 public boolean isKeyType() {
160     return true;
161 }
162
163 @Override
164 public TypeSerializer<T> createSerializer(ExecutionConfig config) {
165     return this.serializer;
166 }
167
168 @Override
169 @PublicEvolving
170 public TypeComparator<T> createComparator(boolean sortOrderAscending, ExecutionConfig executionConfig) {
171     if (comparatorClass != null) {
172         return instantiateComparator(comparatorClass, sortOrderAscending);
173     } else {
174         throw new InvalidTypesException("The type " + clazz.getSimpleName() + " cannot be used as a comparator");
175     }
176 }
```

J BasicTypeInfo.java

```
59 public class PrimitiveColumnInfo<T> extends TypeInformation<T> implements AtomicType<T> {
179 public String toString() {
180     return clazz.getSimpleName();
181 }
182
183 @Override
184 public boolean equals(Object obj) {
185     if (obj instanceof PrimitiveColumnInfo) {
186         @SuppressWarnings("unchecked")
187         PrimitiveColumnInfo<T> other = (PrimitiveColumnInfo<T>) obj;
188
189         return other.canEqual(this) &&
190             this.clazz == other.clazz &&
191             this.serializer.equals(other.serializer) &&
192             this.comparatorClass == other.comparatorClass;
193     } else {
194         return false;
195     }
196 }
197
198 @Override
199 public int hashCode() {
200     return (31 * Objects.hash(clazz, serializer, comparatorClass)) + Arrays.hashCode(possibleComparators);
201 }
202
203 @Override
204 public boolean canEqual(Object obj) {
205     return obj instanceof PrimitiveColumnInfo;
```

```
59 public class BasicTypeInfo<T> extends TypeInformation<T> implements AtomicType<T> {
109 @Override
110 public boolean isBasicType() {
111     return true;
112 }
113
114 @Override
115 public boolean isTupleType() {
116     return false;
117 }
118
119 @Override
120 public int getArity() {
121     return 1;
122 }
123
124 @Override
125 public int getTotalFields() {
126     return 1;
127 }
128
129 @Override
130 public Class<T> getTypeClass() {
131     return this.clazz;
132 }
133
134 @Override
135 public boolean isKeyType() {
136     return true;
137 }
138
139 @Override
140 public TypeSerializer<T> createSerializer(ExecutionConfig executionConfig) {
141     return this.serializer;
142 }
143
144 @Override
145 public TypeComparator<T> createComparator(boolean sortOrderAscending, ExecutionConfig executionConfig) {
146     if (comparatorClass != null) {
147         return instantiateComparator(comparatorClass, sortOrderAscending);
148     } else {
149         throw new InvalidTypesException("The type " + clazz.getSimpleName() + " cannot be used as a comparator");
150     }
151 }
152
153 // -----
154
155 @Override
156 public int hashCode() {
157     return (31 * Objects.hash(clazz, serializer, comparatorClass)) + Arrays.hashCode(possibleComparators);
158 }
159 }
```

```
59 public class BasicTypeInfo<T> extends TypeInformation<T> implements AtomicType<T> {
161+ public boolean canEqual(Object obj) {
162+     return obj instanceof BasicTypeInfo;
163+ }
164
165 @Override
166 public boolean equals(Object obj) {
167+     if (obj instanceof BasicTypeInfo) {
168+         @SuppressWarnings("unchecked")
169+         BasicTypeInfo<T> other = (BasicTypeInfo<T>) obj;
170+
171+         return other.canEqual(this) &&
172+             this.clazz == other.clazz &&
173+             this.serializer.equals(other.serializer) &&
174+             this.comparatorClass == other.comparatorClass;
175+     } else {
176+         return false;
177+     }
178+ }
179
180 @Override
181+ public String toString() {
182+     return clazz.getSimpleName();
183+ }
184
185 // -----
186
187+ public static <X> BasicTypeInfo<X> getFor(Class<X> type) {
188+     if (type == null) {
189+         throw new NullPointerException();
190+     }
191+
192+     @SuppressWarnings("unchecked")
193+     BasicTypeInfo<X> info = (BasicTypeInfo<X>) TYPES.get(type);
194+     return info;
195+ }
196
197+ private static <X> TypeComparator<X> instantiateComparator(Class<?> extends TypeComparator<X>) {
198+     try {
199+         Constructor<?> extends TypeComparator<X> constructor = comparatorClass.getConstructor();
200+         return constructor.newInstance(ascendingOrder);
201+     } catch (Exception e) {
202+         throw new RuntimeException("Could not initialize basic comparator " + comparatorClass.getName());
203+     }
204+ }
205+
206+ private static final Map<Class<?>, BasicTypeInfo<?>> TYPES = new HashMap<Class<?>, BasicTypeInfo<?>>();
207+
208+ static {
209+     TYPES.put(String.class, STRING_TYPE_INFO);
210+     TYPES.put(Boolean.class, BOOLEAN_TYPE_INFO);
211+     TYPES.put(boolean.class, BOOLEAN_TYPE_INFO);
212+     TYPES.put(Byte.class, BYTE_TYPE_INFO);
213+ }
```



### 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.

J AnimatedGifEncoder.java

```
1+ /* *****  
2+  * Copyright (c) 2013 The University of Reading  
3+  * All rights reserved.  
4+  
5+  * Redistribution and use in source and binary forms, with or without  
6+  * modification, are permitted provided that the following conditions  
7+  * are met:  
8+  * 1. Redistributions of source code must retain the above copyright  
9+  * notice, this list of conditions and the following disclaimer.  
10+  * 2. Redistributions in binary form must reproduce the above copyright  
11+  * notice, this list of conditions and the following disclaimer in the  
12+  * documentation and/or other materials provided with the distribution.  
13+  * 3. Neither the name of the University of Reading, nor the names of the  
14+  * authors or contributors may be used to endorse or promote products  
15+  * derived from this software without specific prior written permission.  
16+  
17+  * THIS SOFTWARE IS PROVIDED BY THE AUTHOR ``AS IS'' AND ANY EXPRESS OR  
18+  * IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES  
19+  
20+  * OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED.  
21+  * IN NO EVENT SHALL THE AUTHOR BE LIABLE FOR ANY DIRECT, INDIRECT,  
22+  * INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT  
23+  * NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE,  
24+  * DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY  
25+  * THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT  
26+  * (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF  
27+  * THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.  
28+  *  
29+  * *****/  
30+  
31+ package uk.ac.rdg.resc.edal.graphics.formats;  
32+  
33+  
34+  
35+  
36+  
37+  
38+  
39+  
40+  
41+  
42+  
43+  
44+  
45+  
46+  
47+  
48+  
49+  
50+  
51+  
52+  
53+  
54+  
55+  
56+  
57+  
58+  
59+  
60+  
61+  
62+  
63+  
64+  
65+  
66+  
67+  
68+  
69+  
70+  
71+  
72+  
73+  
74+  
75+  
76+  
77+  
78+  
79+  
80+  
81+  
82+  
83+  
84+  
85+  
86+  
87+  
88+  
89+  
90+  
91+  
92+  
93+  
94+  
95+  
96+  
97+  
98+  
99+  
100+  
101+  
102+  
103+  
104+  
105+  
106+  
107+  
108+  
109+  
110+  
111+  
112+  
113+  
114+  
115+  
116+  
117+  
118+  
119+  
120+  
121+  
122+  
123+  
124+  
125+  
126+  
127+  
128+  
129+  
130+  
131+  
132+  
133+  
134+  
135+  
136+  
137+  
138+  
139+  
140+  
141+  
142+  
143+  
144+  
145+  
146+  
147+  
148+  
149+  
150+  
151+  
152+  
153+  
154+  
155+  
156+  
157+  
158+  
159+  
160+  
161+  
162+  
163+  
164+  
165+  
166+  
167+  
168+  
169+  
170+  
171+  
172+  
173+  
174+  
175+  
176+  
177+  
178+  
179+  
180+  
181+  
182+  
183+  
184+  
185+  
186+  
187+  
188+  
189+  
190+  
191+  
192+  
193+  
194+  
195+  
196+  
197+  
198+  
199+  
200+  
201+  
202+  
203+  
204+  
205+  
206+  
207+  
208+  
209+  
210+  
211+  
212+  
213+  
214+  
215+  
216+  
217+  
218+  
219+  
220+  
221+  
222+  
223+  
224+  
225+  
226+  
227+  
228+  
229+  
230+  
231+  
232+  
233+  
234+  
235+  
236+  
237+  
238+  
239+  
240+  
241+  
242+  
243+  
244+  
245+  
246+  
247+  
248+  
249+  
250+  
251+  
252+  
253+  
254+  
255+  
256+  
257+  
258+  
259+  
260+  
261+  
262+  
263+  
264+  
265+  
266+  
267+  
268+  
269+  
270+  
271+  
272+  
273+  
274+  
275+  
276+  
277+  
278+  
279+  
280+  
281+  
282+  
283+  
284+  
285+  
286+  
287+  
288+  
289+  
290+  
291+  
292+  
293+  
294+  
295+  
296+  
297+  
298+  
299+  
300+  
301+  
302+  
303+  
304+  
305+  
306+  
307+  
308+  
309+  
310+  
311+  
312+  
313+  
314+  
315+  
316+  
317+  
318+  
319+  
320+  
321+  
322+  
323+  
324+  
325+  
326+  
327+  
328+  
329+  
330+  
331+  
332+  
333+  
334+  
335+  
336+  
337+  
338+  
339+  
340+  
341+  
342+  
343+  
344+  
345+  
346+  
347+  
348+  
349+  
350+  
351+  
352+  
353+  
354+  
355+  
356+  
357+  
358+  
359+  
360+  
361+  
362+  
363+  
364+  
365+  
366+  
367+  
368+  
369+  
370+  
371+  
372+  
373+  
374+  
375+  
376+  
377+  
378+  
379+  
380+  
381+  
382+  
383+  
384+  
385+  
386+  
387+  
388+  
389+  
390+  
391+  
392+  
393+  
394+  
395+  
396+  
397+  
398+  
399+  
400+  
401+  
402+  
403+  
404+  
405+  
406+  
407+  
408+  
409+  
410+  
411+  
412+  
413+  
414+  
415+  
416+  
417+  
418+  
419+  
420+  
421+  
422+  
423+  
424+  
425+  
426+  
427+  
428+  
429+  
430+  
431+  
432+  
433+  
434+  
435+  
436+  
437+  
438+  
439+  
440+  
441+  
442+  
443+  
444+  
445+  
446+  
447+  
448+  
449+  
450+  
451+  
452+  
453+  
454+  
455+  
456+  
457+  
458+  
459+  
460+  
461+  
462+  
463+  
464+  
465+  
466+  
467+  
468+  
469+  
470+  
471+  
472+  
473+  
474+  
475+  
476+  
477+  
478+  
479+  
480+  
481+  
482+  
483+  
484+  
485+  
486+  
487+  
488+  
489+  
490+  
491+  
492+  
493+  
494+  
495+  
496+  
497+  
498+  
499+  
500+  
501+  
502+  
503+  
504+  
505+  
506+  
507+  
508+  
509+  
510+  
511+  
512+  
513+  
514+  
515+  
516+  
517+  
518+  
519+  
520+  
521+  
522+  
523+  
524+  
525+  
526+  
527+  
528+  
529+  
530+  
531+  
532+  
533+  
534+  
535+  
536+  
537+  
538+  
539+  
540+  
541+  
542+  
543+  
544+  
545+  
546+  
547+  
548+  
549+  
550+  
551+  
552+  
553+  
554+  
555+  
556+  
557+  
558+  
559+  
560+  
561+  
562+  
563+  
564+  
565+  
566+  
567+  
568+  
569+  
570+  
571+  
572+  
573+  
574+  
575+  
576+  
577+  
578+  
579+  
580+  
581+  
582+  
583+  
584+  
585+  
586+  
587+  
588+  
589+  
590+  
591+  
592+  
593+  
594+  
595+  
596+  
597+  
598+  
599+  
600+  
601+  
602+  
603+  
604+  
605+  
606+  
607+  
608+  
609+  
610+  
611+  
612+  
613+  
614+  
615+  
616+  
617+  
618+  
619+  
620+  
621+  
622+  
623+  
624+  
625+  
626+  
627+  
628+  
629+  
630+  
631+  
632+  
633+  
634+  
635+  
636+  
637+  
638+  
639+  
640+  
641+  
642+  
643+  
644+  
645+  
646+  
647+  
648+  
649+  
650+  
651+  
652+  
653+  
654+  
655+  
656+  
657+  
658+  
659+  
660+  
661+  
662+  
663+  
664+  
665+  
666+  
667+  
668+  
669+  
670+  
671+  
672+  
673+  
674+  
675+  
676+  
677+  
678+  
679+  
680+  
681+  
682+  
683+  
684+  
685+  
686+  
687+  
688+  
689+  
690+  
691+  
692+  
693+  
694+  
695+  
696+  
697+  
698+  
699+  
700+  
701+  
702+  
703+  
704+  
705+  
706+  
707+  
708+  
709+  
710+  
711+  
712+  
713+  
714+  
715+  
716+  
717+  
718+  
719+  
720+  
721+  
722+  
723+  
724+  
725+  
726+  
727+  
728+  
729+  
730+  
731+  
732+  
733+  
734+  
735+  
736+  
737+  
738+  
739+  
740+  
741+  
742+  
743+  
744+  
745+  
746+  
747+  
748+  
749+  
750+  
751+  
752+  
753+  
754+  
755+  
756+  
757+  
758+  
759+  
760+  
761+  
762+  
763+  
764+  
765+  
766+  
767+  
768+  
769+  
770+  
771+  
772+  
773+  
774+  
775+  
776+  
777+  
778+  
779+  
780+  
781+  
782+  
783+  
784+  
785+  
786+  
787+  
788+  
789+  
790+  
791+  
792+  
793+  
794+  
795+  
796+  
797+  
798+  
799+  
800+  
801+  
802+  
803+  
804+  
805+  
806+  
807+  
808+  
809+  
810+  
811+  
812+  
813+  
814+  
815+  
816+  
817+  
818+  
819+  
820+  
821+  
822+  
823+  
824+  
825+  
826+  
827+  
828+  
829+  
830+  
831+  
832+  
833+  
834+  
835+  
836+  
837+  
838+  
839+  
840+  
841+  
842+  
843+  
844+  
845+  
846+  
847+  
848+  
849+  
850+  
851+  
852+  
853+  
854+  
855+  
856+  
857+  
858+  
859+  
860+  
861+  
862+  
863+  
864+  
865+  
866+  
867+  
868+  
869+  
870+  
871+  
872+  
873+  
874+  
875+  
876+  
877+  
878+  
879+  
880+  
881+  
882+  
883+  
884+  
885+  
886+  
887+  
888+  
889+  
890+  
891+  
892+  
893+  
894+  
895+  
896+  
897+  
898+  
899+  
900+  
901+  
902+  
903+  
904+  
905+  
906+  
907+  
908+  
909+  
910+  
911+  
912+  
913+  
914+  
915+  
916+  
917+  
918+  
919+  
920+  
921+  
922+  
923+  
924+  
925+  
926+  
927+  
928+  
929+  
930+  
931+  
932+  
933+  
934+  
935+  
936+  
937+  
938+  
939+  
940+  
941+  
942+  
943+  
944+  
945+  
946+  
947+  
948+  
949+  
950+  
951+  
952+  
953+  
954+  
955+  
956+  
957+  
958+  
959+  
960+  
961+  
962+  
963+  
964+  
965+  
966+  
967+  
968+  
969+  
970+  
971+  
972+  
973+  
974+  
975+  
976+  
977+  
978+  
979+  
980+  
981+  
982+  
983+  
984+  
985+  
986+  
987+  
988+  
989+  
990+  
991+  
992+  
993+  
994+  
995+  
996+  
997+  
998+  
999+  
1000+  
1001+  
1002+  
1003+  
1004+  
1005+  
1006+  
1007+  
1008+  
1009+  
1010+  
1011+  
1012+  
1013+  
1014+  
1015+  
1016+  
1017+  
1018+  
1019+  
1020+  
1021+  
1022+  
1023+  
1024+  
1025+  
1026+  
1027+  
1028+  
1029+  
1030+  
1031+  
1032+  
1033+  
1034+  
1035+  
1036+  
1037+  
1038+  
1039+  
1040+  
1041+  
1042+  
1043+  
1044+  
1045+  
1046+  
1047+  
1048+  
1049+  
1050+  
1051+  
1052+  
1053+  
1054+  
1055+  
1056+  
1057+  
1058+  
1059+  
1060+  
1061+  
1062+  
1063+  
1064+  
1065+  
1066+  
1067+  
1068+  
1069+  
1070+  
1071+  
1072+  
1073+  
1074+  
1075+  
1076+  
1077+  
1078+  
1079+  
1080+  
1081+  
1082+  
1083+  
1084+  
1085+  
1086+  
1087+  
1088+  
1089+  
1090+  
1091+  
1092+  
1093+  
1094+  
1095+  
1096+  
1097+  
1098+  
1099+  
1100+  
1101+  
1102+  
1103+  
1104+  
1105+  
1106+  
1107+  
1108+  
1109+  
1110+  
1111+  
1112+  
1113+  
1114+  
1115+  
1116+  
1117+  
1118+  
1119+  
1120+  
1121+  
1122+  
1123+  
1124+  
1125+  
1126+  
1127+  
1128+  
1129+  
1130+  
1131+  
1132+  
1133+  
1134+  
1135+  
1136+  
1137+  
1138+  
1139+  
1140+  
1141+  
1142+  
1143+  
1144+  
1145+  
1146+  
1147+  
1148+  
1149+  
1150+  
1151+  
1152+  
1153+  
1154+  
1155+  
1156+  
1157+  
1158+  
1159+  
1160+  
1161+  
1162+  
1163+  
1164+  
1165+  
1166+  
1167+  
1168+  
1169+  
1170+  
1171+  
1172+  
1173+  
1174+  
1175+  
1176+  
1177+  
1178+  
1179+  
1180+  
1181+  
1182+  
1183+  
1184+  
1185+  
1186+  
1187+  
1188+  
1189+  
1190+  
1191+  
1192+  
1193+  
1194+  
1195+  
1196+  
1197+  
1198+  
1199+  
1200+  
1201+  
1202+  
1203+  
1204+  
1205+  
1206+  
1207+  
1208+  
1209+  
1210+  
1211+  
1212+  
1213+  
1214+  
1215+  
1216+  
1217+  
1218+  
1219+  
1220+  
1221+  
1222+  
1223+  
1224+  
1225+  
1226+  
1227+  
1228+  
1229+  
1230+  
1231+  
1232+  
1233+  
1234+  
1235+  
1236+  
1237+  
1238+  
1239+  
1240+  
1241+  
1242+  
1243+  
1244+  
1245+  
1246+  
1247+  
1248+  
1249+  
1250+  
1251+  
1252+  
1253+  
1254+  
1255+  
1256+  
1257+  
1258+  
1259+  
1260+  
1261+  
1262+  
1263+  
1264+  
1265+  
1266+  
1267+  
1268+  
1269+  
1270+  
1271+  
1272+  
1273+  
1274+  
1275+  
1276+  
1277+  
1278+  
1279+  
1280+  
1281+  
1282+  
1283+  
1284+  
1285+  
1286+  
1287+  
1288+  
1289+  
1290+  
1291+  
1292+  
1293+  
1294+  
1295+  
1296+  
1297+  
1298+  
1299+  
1300+  
1301+  
1302+  
1303+  
1304+  
1305+  
1306+  
1307+  
1308+  
1309+  
1310+  
1311+  
1312+  
1313+  
1314+  
1315+  
1316+  
1317+  
1318+  
1319+  
1320+  
1321+  
1322+  
1323+  
1324+  
1325+  
1326+  
1327+  
1328+  
1329+  
1330+  
1331+  
1332+  
1333+  
1334+  
1335+  
1336+  
1337+  
1338+  
1339+  
1340+  
1341+  
1342+  
1343+  
1344+  
1345+  
1346+  
1347+  
1348+  
1349+  
1350+  
1351+  
1352+  
1353+  
1354+  
1355+  
1356+  
1357+  
1358+  
1359+  
1360+  
1361+  
1362+  
1363+  
1364+  
1365+  
1366+  
1367+  
1368+  
1369+  
1370+  
1371+  
1372+  
1373+  
1374+  
1375+  
1376+  
1377+  
1378+  
1379+  
1380+  
1381+  
1382+  
1383+  
1384+  
1385+  
1386+  
1387+  
1388+  
1389+  
1390+  
1391+  
1392+  
1393+  
1394+  
1395+  
1396+  
1397+  
1398+  
1399+  
1400+  
1401+  
1402+  
1403+  
1404+  
1405+  
1406+  
1407+  
1408+  
1409+  
1410+  
1411+  
1412+  
1413+  
1414+  
1415+  
1416+  
1417+  
1418+  
1419+  
1420+  
1421+  
1422+  
1423+  
1424+  
1425+  
1426+  
1427+  
1428+  
1429+  
1430+  
1431+  
1432+  
1433+  
1434+  
1435+  
1436+  
1437+  
1438+  
1439+  
1440+  
1441+  
1442+  
1443+  
1444+  
1445+  
1446+  
1447+  
1448+  
1449+  
1450+  
1451+  
1452+  
1453+  
1454+  
1455+  
1456+  
1457+  
1458+  
1459+  
1460+  
1461+  
1462+  
1463+  
1464+  
1465+  
1466+  
1467+  
1468+  
1469+  
1470+  
1471+  
1472+  
1473+  
1474+  
1475+  
1476+  
1477+  
1478+  
1479+  
1480+  
1481+  
1482+  
1483+  
1484+  
1485+  
1486+  
1487+  
1488+  
1489+  
1490+  
1491+  
1492+  
1493+  
1494+  
1495+  
1496+  
1497+  
1498+  
1499+  
1500+  
1501+  
1502+  
1503+  
1504+  
1505+  
1506+  
1507+  
1508+  
1509+  
1510+  
1511+  
1512+  
1513+  
1514+  
1515+  
1516+  
1517+  
1518+  
1519+  
1520+  
1521+  
1522+  
1523+  
1524+  
1525+  
1526+  
1527+  
1528+  
1529+  
1530+  
1531+  
1532+  
1533+  
1534+  
1535+  
1536+  
1537+  
1538+  
1539+  
1540+  
1541+  
1542+  
1543+  
1544+  
1545+  
1546+  
1547+  
1548+  
1549+  
1550+  
1551+  
1552+  
1553+  
1554+  
1555+  
1556+  
1557+  
1558+  
1559+  
1560+  
1561+  
1562+  
1563+  
1564+  
1565+  
1566+  
1567+  
1568+  
1569+  
1570+  
1571+  
1572+  
1573+  
1574+  
1575+  
1576+  
1577+  
1578+  
1579+  
1580+  
1581+  
1582+  
1583+  
1584+  
1585+  
1586+  
1587+  
1588+  
1589+  
1590+  
1591+  
1592+  
1593+  
1594+  
1595+  
1596+  
1597+  
1598+  
1599+  
1600+  
1601+  
1602+  
1603+  
1604+  
1605+  
1606+  
1607+  
1608+  
1609+  
1610+  
1611+  
1612+  
1613+  
1614+  
1615+  
1616+  
1617+  
1618+  
1619+  
1620+  
1621+  
1622+  
1623+  
1624+  
1625+  
1626+  
1627+  
1628+  
1629+  
1630+  
1631+  
1632+  
1633+  
1634+  
1635+  
1636+  
1637+  
1638+  
1639+  
16
```

J AnimatedGifEncoder.java

C:\Users\86969\Downloads\JavaCloneResult\AnimatedGifEncoder.java

```
35 // Adapted from Jef Poskanzen's Java port by way of J. M. G. Elliott.  
36 // K Weiner 12/00
```

```
37  
38 class LZWEncoder {  
39  
40     private static final int EOF = -1;  
41  
42     private int imgW, imgH;  
43  
44     private byte[] pixAry;  
45  
46     private int initCodeSize;  
47  
48     private int remaining;  
49  
50     private int curPixel;  
51  
52     // GIFCOMP.R - GIF Image compression routines  
53     //  
54     // Lempel-Ziv compression based on 'compress'. GIF modifications by  
55     // David Rowley (ngardl@watdcsu.waterloo.edu)  
56  
57     // General DEFINES  
58  
59     static final int BITS = 12;  
60  
61     static final int HSIZE = 5003; // 80% occupancy  
62  
63     // GIF Image compression - modified 'compress'  
64     //  
65     // Based on: compress.c - File compression ala IEEE Computer, June 1984.  
66  
67     // By Authors: Spencer W. Thomas (decvax!harpoluth-csl.utah-gr!thomas)  
68     //              Jim McKie (decvax!mcvax!jim)  
69     //              Steve Davies (decvax!vax135!petsd!peora!srd)  
70     //              Ken Turkowski (decvax!decw!turt!levax!ken)  
71     //              James A. Woods (decvax!ihnp4!ames!jaw)  
72     //              Joe Orost (decvax!vax135!petsd!joe)
```

```
73     int n_bits; // number of bits/code  
74  
75     int maxbits = BITS; // user settable max # bits/code  
76  
77     int maxcode; // maximum code, given n_bits
```

J AnimatedGifEncoder.java

```
38 class LZWEncoder {  
39     int maxcode; // maximum code, given n_bits  
40  
41     int maxmaxcode = 1 << BITS; // should NEVER generate this code  
42  
43     int[] htab = new int[HSIZE];  
44  
45     int[] codetab = new int[HSIZE];  
46  
47     int hsize = HSIZE; // for dynamic table sizing  
48  
49     int free_ent = 0; // first unused entry  
50  
51     // block compression parameters -- after all codes are used up,  
52     // and compression rate changes, start over.  
53     boolean clear_flg = false;  
54  
55     // Algorithm: use open addressing double hashing (no chaining) on the  
56     // prefix code / next character combination. We do a variant of Knuth's  
57     // algorithm D (vol. 3, sec. 6.4) along with G. Knott's relatively-prime  
58     // secondary probe. Here, the modular division first probe is gives way  
59     // to a faster exclusive-or manipulation. Also do block compression with  
60     // an adaptive reset, whereby the code table is cleared when the compression  
61     // ratio decreases, but after the table fills. The variable-length output  
62     // codes are re-sized at this point, and a special CLEAR code is generated  
63     // for the decompressor. Late addition: construct the table according to  
64     // file size for noticeable speed improvement on small files. Please direct  
65     // questions about this implementation to ames!jaw.  
66  
67     int g_init_bits;  
68  
69     int ClearCode;  
70  
71     int EOFCode;  
72  
73     // output  
74     //  
75     // Output the given code.  
76     // Inputs:  
77     //     code: A n_bits-bit integer. If == -1, then EOF. This assumes  
78     //     that n_bits <= wordsize - 1.  
79     // Outputs:  
80     //     Outputs code to the file.  
81     // Assumptions:  
82     //     Chars are 8 bits long.  
83     // Algorithm:  
84     //     Maintain a BITS character long buffer (so that 8 codes will  
85     //     fit in it exactly). Use the VAX insv instruction to insert each  
86     //     code in turn. When the buffer fills up empty it and start over.  
87  
88     int cur_accum = 0;  
89  
90     int cur_bits = 0;  
91  
92     int masks[] = { 0x0000, 0x0001, 0x0003, 0x0007, 0x000F, 0x001F, 0x003F, 0x007F, 0x00FF, 0x01FF, 0x03FF, 0x07FF, 0x0FFF, 0x1FFF, 0x3FFF, 0x7FFF, 0xFFFF }
```

```
558 class NeuQuant {  
559     protected int contest(int b, int g, int r) {  
560         freq[bestpos] += beta;  
561         bias[bestpos] -= betagamma;  
562         return (bestbiaspos);  
563     }  
564 }  
565
```

```
1045 // Adapted from Jef Poskanzen's Java port by way of J. M. G. Elliott.  
1046 // K Weiner 12/00
```

```
1048 class LZWEncoder {  
1049  
1050     private static final int EOF = -1;  
1051  
1052     private int imgW, imgH;  
1053  
1054     private byte[] pixAry;  
1055  
1056     private int initCodeSize;  
1057  
1058     private int remaining;  
1059  
1060     private int curPixel;  
1061  
1062     // GIFCOMP.R - GIF Image compression routines  
1063     //  
1064     // Lempel-Ziv compression based on 'compress'. GIF modifications by  
1065     // David Rowley (ngardl@watdcsu.waterloo.edu)  
1066  
1067     // General DEFINES  
1068  
1069     static final int BITS = 12;  
1070  
1071     static final int HSIZE = 5003; // 80% occupancy  
1072  
1073     // GIF Image compression - modified 'compress'  
1074     //  
1075     // Based on: compress.c - File compression ala IEEE Computer, June 1984.  
1076  
1077     // By Authors: Spencer W. Thomas (decvax!harpoluth-csl.utah-gr!thomas)  
1078     //              Jim McKie (decvax!mcvax!jim)  
1079     //              Steve Davies (decvax!vax135!petsd!peora!srd)  
1080     //              Ken Turkowski (decvax!decw!turt!levax!ken)  
1081     //              James A. Woods (decvax!ihnp4!ames!jaw)  
1082     //              Joe Orost (decvax!vax135!petsd!joe)
```

```
1083     int n_bits; // number of bits/code  
1084  
1085     int maxbits = BITS; // user settable max # bits/code  
1086  
1087     int maxcode; // maximum code, given n_bits
```

```
1048 class LZWEncoder {  
1049     int maxcode; // maximum code, given n_bits  
1050  
1051     int maxmaxcode = 1 << BITS; // should NEVER generate this code  
1052  
1053     int[] htab = new int[HSIZE];  
1054  
1055     int[] codetab = new int[HSIZE];  
1056  
1057     int hsize = HSIZE; // for dynamic table sizing  
1058  
1059     int free_ent = 0; // first unused entry  
1060  
1061     // block compression parameters -- after all codes are used up,  
1062     // and compression rate changes, start over.  
1063     boolean clear_flg = false;  
1064  
1065     // Algorithm: use open addressing double hashing (no chaining) on the  
1066     // prefix code / next character combination. We do a variant of Knuth's  
1067     // algorithm D (vol. 3, sec. 6.4) along with G. Knott's relatively-prime  
1068     // secondary probe. Here, the modular division first probe is gives way  
1069     // to a faster exclusive-or manipulation. Also do block compression with  
1070     // an adaptive reset, whereby the code table is cleared when the compression  
1071     // ratio decreases, but after the table fills. The variable-length output  
1072     // codes are re-sized at this point, and a special CLEAR code is generated  
1073     // for the decompressor. Late addition: construct the table according to  
1074     // file size for noticeable speed improvement on small files. Please direct  
1075     // questions about this implementation to ames!jaw.  
1076  
1077     int g_init_bits;  
1078  
1079     int ClearCode;  
1080  
1081     int EOFCode;  
1082  
1083     // output  
1084     //  
1085     // Output the given code.  
1086     // Inputs:  
1087     //     code: A n_bits-bit integer. If == -1, then EOF. This assumes  
1088     //     that n_bits <= wordsize - 1.  
1089     // Outputs:  
1090     //     Outputs code to the file.  
1091     // Assumptions:  
1092     //     Chars are 8 bits long.  
1093     // Algorithm:  
1094     //     Maintain a BITS character long buffer (so that 8 codes will  
1095     //     fit in it exactly). Use the VAX insv instruction to insert each  
1096     //     code in turn. When the buffer fills up empty it and start over.  
1097  
1098     int cur_accum = 0;  
1099  
1100     int cur_bits = 0;  
1101  
1102     int masks[] = { 0x0000, 0x0001, 0x0003, 0x0007, 0x000F, 0x001F, 0x003F, 0x007F, 0x00FF, 0x01FF, 0x03FF, 0x07FF, 0x0FFF, 0x1FFF, 0x3FFF, 0x7FFF, 0xFFFF }
```

J AnimatedGifEncoder.java

```
38 class LZLEncoder {
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 LZLEncoder(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
```

J AnimatedGifEncoder.java

```
38 class LZLEncoder {
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;
227     }
228 }
```

```
1048 class LZLEncoder {
1141- int masks[] = { 0x0000, 0x0001, 0x0003, 0x0007, 0x000F, 0x001F, 0x003F, 0x007F, 0x00FF, 0x01FF,
1142-                0x03FF, 0x07FF, 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 LZLEncoder(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
```

```
1048 class LZLEncoder {
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;
1230     }
1231 }
```



J AnimatedGifEncoder.java

```
38 class LZWEncoder {
180 void compress(int init_bits, OutputStream outs) throws IOException {
215 outer_loop: while ((c = nextPixel()) != EOF) {
222 } else if (htab[i] >= 0) // non-empty slot
223 do {
230
231     if (htab[i] == fcode) {
232         ent = codetab[i];
233         continue outer_loop;
234     }
235     } while (htab[i] >= 0);
236
237     output(ent, outs);
238     ent = c;
239     if (free_ent < maxmaxcode) {
240         codetab[i] = free_ent++; // code -> hashtable
241         htab[i] = fcode;
242     } else
243         cl_block(outs);
244 }
245 // Put out the final code.
246 output(ent, outs);
247 output(EOFCode, outs);
248 }
249
250 //-----
251 void encode(OutputStream os) throws IOException {
252     os.write(initCodeSize); // write "initial code size" byte
253
254     remaining = imgW * imgH; // reset navigation variables
255     curPixel = 0;
256
257     compress(initCodeSize + 1, os); // compress and write the pixel data
258
259     os.write(0); // write block terminator
260 }
261
262 // Flush the packet to disk, and reset the accumulator
263 void flush_char(OutputStream outs) throws IOException {
264     if (a_count > 0) {
265         outs.write(a_count);
266         outs.write(accum, 0, a_count);
267         a_count = 0;
268     }
269 }
270
271 final int MAXCODE(int n_bits) {
272     return (1 << n_bits) - 1;
273 }
274
275 //-----
276 // Return the next pixel from the image
277 private int nextPixel() {
278     if (remaining == 0)
279         return -1;
280 }
```

J AnimatedGifEncoder.java

```
38 class LZWEncoder {
278 private int nextPixel() {
279     if (remaining == 0)
280         return -1;
281
282     --remaining;
283
284     byte pix = pixAry[curPixel++];
285
286     return pix & 0xff;
287 }
288
289 void output(int code, OutputStream outs) throws IOException {
290     cur_accum &= masks[cur_bits];
291
292     if (cur_bits > 0)
293         cur_accum |= (code << cur_bits);
294     else
295         cur_accum = code;
296
297     cur_bits += n_bits;
298
299     while (cur_bits >= 8) {
300         char_out((byte) (cur_accum & 0xff), outs);
301         cur_accum >>= 8;
302         cur_bits -= 8;
303     }
304
305     // If the next entry is going to be too big for the code size,
306     // then increase it, if possible.
307     if (free_ent > maxcode || clear_flg) {
308         if (clear_flg) {
309             maxcode = MAXCODE(n_bits - g_init_bits);
310             clear_flg = false;
311         } else {
312             ++n_bits;
313             if (n_bits == maxbits)
314                 maxcode = maxmaxcode;
315             else
316                 maxcode = MAXCODE(n_bits);
317         }
318     }
319
320     if (code == EOFCode) {
321         // At EOF, write the rest of the buffer.
322         while (cur_bits > 0) {
323             char_out((byte) (cur_accum & 0xff), outs);
324             cur_accum >>= 8;
325             cur_bits -= 8;
326         }
327
328         flush_char(outs);
329     }
330 }
331 }
```

```
1048 class LZWEncoder {
1183 void compress(int init_bits, OutputStream outs) throws IOException {
1218 outer_loop: while ((c = nextPixel()) != EOF) {
1226 }
1230 do {
1233
1234     if (htab[i] == fcode) {
1235         ent = codetab[i];
1236         continue outer_loop;
1237     }
1238     } while (htab[i] >= 0);
1239
1240     output(ent, outs);
1241     ent = c;
1242     if (free_ent < maxmaxcode) {
1243         codetab[i] = free_ent++; // code -> hashtable
1244         htab[i] = fcode;
1245     } else
1246         cl_block(outs);
1247 }
1248 // Put out the final code.
1249 output(ent, outs);
1250 output(EOFCode, outs);
1251 }
1252
1253 //-----
1254 void encode(OutputStream os) throws IOException {
1255     os.write(initCodeSize); // write "initial code size" byte
1256
1257     remaining = imgW * imgH; // reset navigation variables
1258     curPixel = 0;
1259
1260     compress(initCodeSize + 1, os); // compress and write the pixel data
1261
1262     os.write(0); // write block terminator
1263 }
1264
1265 // Flush the packet to disk, and reset the accumulator
1266 void flush_char(OutputStream outs) throws IOException {
1267     if (a_count > 0) {
1268         outs.write(a_count);
1269         outs.write(accum, 0, a_count);
1270         a_count = 0;
1271     }
1272 }
1273
1274 final int MAXCODE(int n_bits) {
1275     return (1 << n_bits) - 1;
1276 }
1277
1278 //-----
1279 // Return the next pixel from the image
1280 private int nextPixel() {
1281     if (remaining == 0)
1282         return -1;
1283 }
```

```
1048 class LZWEncoder {
1281 private int nextPixel() {
1282     if (remaining == 0)
1283         return -1;
1284
1285     --remaining;
1286
1287     byte pix = pixAry[curPixel++];
1288
1289     return pix & 0xff;
1290 }
1291
1292 void output(int code, OutputStream outs) throws IOException {
1293     cur_accum &= masks[cur_bits];
1294
1295     if (cur_bits > 0)
1296         cur_accum |= (code << cur_bits);
1297     else
1298         cur_accum = code;
1299
1300     cur_bits += n_bits;
1301
1302     while (cur_bits >= 8) {
1303         char_out((byte) (cur_accum & 0xff), outs);
1304         cur_accum >>= 8;
1305         cur_bits -= 8;
1306     }
1307
1308     // If the next entry is going to be too big for the code size,
1309     // then increase it, if possible.
1310     if (free_ent > maxcode || clear_flg) {
1311         if (clear_flg) {
1312             maxcode = MAXCODE(n_bits - g_init_bits);
1313             clear_flg = false;
1314         } else {
1315             ++n_bits;
1316             if (n_bits == maxbits)
1317                 maxcode = maxmaxcode;
1318             else
1319                 maxcode = MAXCODE(n_bits);
1320         }
1321     }
1322
1323     if (code == EOFCode) {
1324         // At EOF, write the rest of the buffer.
1325         while (cur_bits > 0) {
1326             char_out((byte) (cur_accum & 0xff), outs);
1327             cur_accum >>= 8;
1328             cur_bits -= 8;
1329         }
1330
1331         flush_char(outs);
1332     }
1333 }
1334 }
```