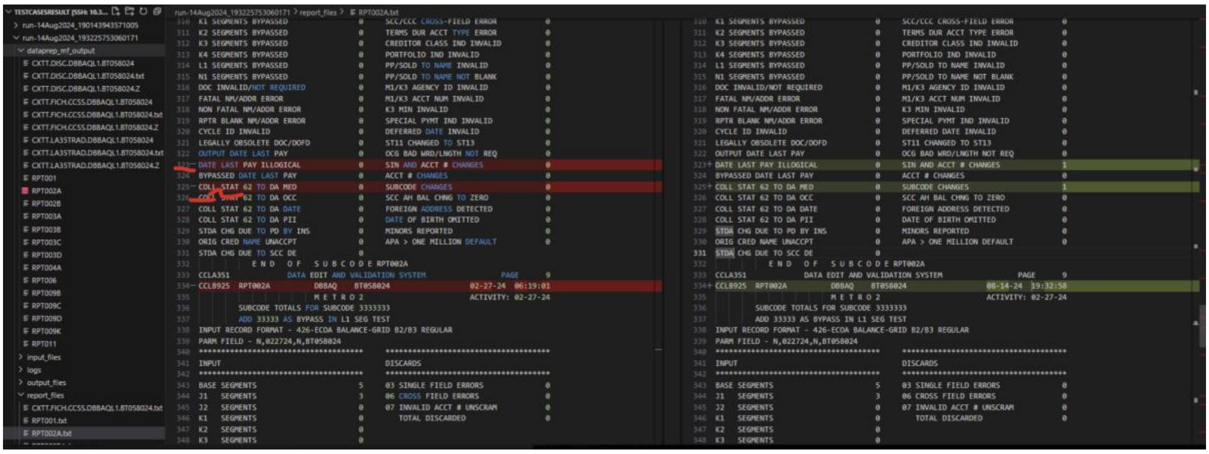
## Problem Statement

There's a discrepancy in the output between the mainframe (MF) code and the AWS implementation:

- **MF output:** COLL STAT 62 TO DA MED 0 SUBCODE CHANGES 0
- **AWS output:** COLL STAT 62 TO DA MED 0 SUBCODE CHANGES 1

The difference is in the SUBCODE CHANGES value, which is 0 on MF and 1 on AWS.


TESTCASESRESULT [SSH: 10.3...
run-14Aug2024_190143943571005
run-14Aug2024_193225753060171
dataprep_mf_output
CXTT.DISC.DBBAQL1.BT058024
CXTT.DISC.DBBAQL1.BT058024.txt
CXTT.DISC.DBBAQL1.BT058024.Z
CXTT.FICH.CCSS.DBBAQL1.BT058024
CXTT.FICH.CCSS.DBBAQL1.BT058024.txt
CXTT.FICH.CCSS.DBBAQL1.BT058024.Z
CXTT.LA35TRAD.DBBAQL1.BT058024
CXTT.LA35TRAD.DBBAQL1.BT058024.txt
CXTT.LA35TRAD.DBBAQL1.BT058024.Z
RPT001
RPT002A
RPT002B
RPT003A
RPT003B
RPT003C
RPT003D
RPT004A
RPT006
RPT009B
RPT009C
RPT009D
RPT009K
RPT011
input_files
logs
output_files
report_files
CXTT.FICH.CCSS.DBBAQL1.BT058024.txt
RPT001.txt
RPT002A.txt
run-14Aug2024_193225753060171 > report_files > RPT002A.txt

```
K1 SEGMENTS BYPASSED                0     SCC/CCC CROSS-FIELD ERROR          0
K2 SEGMENTS BYPASSED                0     TERMS DUR ACCT TYPE ERROR          0
K3 SEGMENTS BYPASSED                0     CREDITOR CLASS IND INVALID         0
K4 SEGMENTS BYPASSED                0     PORTFOLIO IND INVALID              0
L1 SEGMENTS BYPASSED                0     PP/SOLD TO NAME INVALID            0
N1 SEGMENTS BYPASSED                0     PP/SOLD TO NAME NOT BLANK          0
DOC INVALID/NOT REQUIRED            0     M1/K3 AGENCY ID INVALID            0
FATAL NM/ADDR ERROR                 0     M1/K3 ACCT NUM INVALID             0
NON FATAL NM/ADDR ERROR             0     K3 MIN INVALID                     0
RPTR BLANK NM/ADDR ERROR            0     SPECIAL PYMT IND INVALID           0
CYCLE ID INVALID                    0     DEFERRED DATE INVALID              0
LEGALLY OBSOLETE DOC/DOFD           0     ST11 CHANGED TO ST13               0
OUTPUT DATE LAST PAY                0     OCG BAD WRD/LNGTH NOT REQ          0
DATE LAST PAY ILLOGICAL             0     SIN AND ACCT # CHANGES             0
BYPASSED DATE LAST PAY              0     ACCT # CHANGES                     0
COLL STAT 62 TO DA MED              0     SUBCODE CHANGES                    0
COLL STAT 62 TO DA OCC              0     SCC AH BAL CHNG TO ZERO            0
COLL STAT 62 TO DA DATE             0     FOREIGN ADDRESS DETECTED           0
COLL STAT 62 TO DA PII              0     DATE OF BIRTH OMITTED              0
STDA CHG DUE TO PD BY INS           0     MINORS REPORTED                    0
ORIG CRED NAME UNACCPT              0     APA > ONE MILLION DEFAULT          0
STDA CHG DUE TO SCC DE              0
          E N D   O F   S U B C O D E RPT002A
CCLA351           DATA EDIT AND VALIDATION SYSTEM              PAGE      9
CCLB925  RPT002A          DBBAQ    BT058024             02-27-24  06:19:01
                          M E T R O 2                   ACTIVITY: 02-27-24
            SUBCODE TOTALS FOR SUBCODE 3333333
            ADD 33333 AS BYPASS IN L1 SEG TEST
INPUT RECORD FORMAT - 426-ECOA BALANCE-GRID B2/B3 REGULAR
PARM FIELD - N,022724,N,BT058024
************************************     ************************************
INPUT                                     DISCARDS
************************************     ************************************
BASE SEGMENTS                       5     03 SINGLE FIELD ERRORS             0
J1   SEGMENTS                       3     06 CROSS FIELD ERRORS              0
J2   SEGMENTS                       0     07 INVALID ACCT # UNSCRAM          0
K1   SEGMENTS                       0        TOTAL DISCARDED                 0
K2   SEGMENTS                       0
K3   SEGMENTS                       0
```

```
K1 SEGMENTS BYPASSED                0     SCC/CCC CROSS-FIELD ERROR          0
K2 SEGMENTS BYPASSED                0     TERMS DUR ACCT TYPE ERROR          0
K3 SEGMENTS BYPASSED                0     CREDITOR CLASS IND INVALID         0
K4 SEGMENTS BYPASSED                0     PORTFOLIO IND INVALID              0
L1 SEGMENTS BYPASSED                0     PP/SOLD TO NAME INVALID            0
N1 SEGMENTS BYPASSED                0     PP/SOLD TO NAME NOT BLANK          0
DOC INVALID/NOT REQUIRED            0     M1/K3 AGENCY ID INVALID            0
FATAL NM/ADDR ERROR                 0     M1/K3 ACCT NUM INVALID             0
NON FATAL NM/ADDR ERROR             0     K3 MIN INVALID                     0
RPTR BLANK NM/ADDR ERROR            0     SPECIAL PYMT IND INVALID           0
CYCLE ID INVALID                    0     DEFERRED DATE INVALID              0
LEGALLY OBSOLETE DOC/DOFD           0     ST11 CHANGED TO ST13               0
OUTPUT DATE LAST PAY                0     OCG BAD WRD/LNGTH NOT REQ          0
DATE LAST PAY ILLOGICAL             0     SIN AND ACCT # CHANGES             1
BYPASSED DATE LAST PAY              0     ACCT # CHANGES                     0
COLL STAT 62 TO DA MED              0     SUBCODE CHANGES                    1
COLL STAT 62 TO DA OCC              0     SCC AH BAL CHNG TO ZERO            0
COLL STAT 62 TO DA DATE             0     FOREIGN ADDRESS DETECTED           0
COLL STAT 62 TO DA PII              0     DATE OF BIRTH OMITTED              0
STDA CHG DUE TO PD BY INS           0     MINORS REPORTED                    0
ORIG CRED NAME UNACCPT              0     APA > ONE MILLION DEFAULT          0
STDA CHG DUE TO SCC DE              0
          E N D   O F   S U B C O D E RPT002A
CCLA351           DATA EDIT AND VALIDATION SYSTEM              PAGE      9
CCLB925  RPT002A          DBBAQ    BT058024             08-14-24  19:32:58
                          M E T R O 2                   ACTIVITY: 02-27-24
            SUBCODE TOTALS FOR SUBCODE 3333333
            ADD 33333 AS BYPASS IN L1 SEG TEST
INPUT RECORD FORMAT - 426-ECOA BALANCE-GRID B2/B3 REGULAR
PARM FIELD - N,022724,N,BT058024
************************************     ************************************
INPUT                                     DISCARDS
************************************     ************************************
BASE SEGMENTS                       5     03 SINGLE FIELD ERRORS             0
J1   SEGMENTS                       3     06 CROSS FIELD ERRORS              0
J2   SEGMENTS                       0     07 INVALID ACCT # UNSCRAM          0
K1   SEGMENTS                       0        TOTAL DISCARDED                 0
K2   SEGMENTS                       0
K3   SEGMENTS                       0
```

## Analysis Steps

The code provided contains everything needed to identify and resolve the issue. Your task is to determine where the logic needs modification.

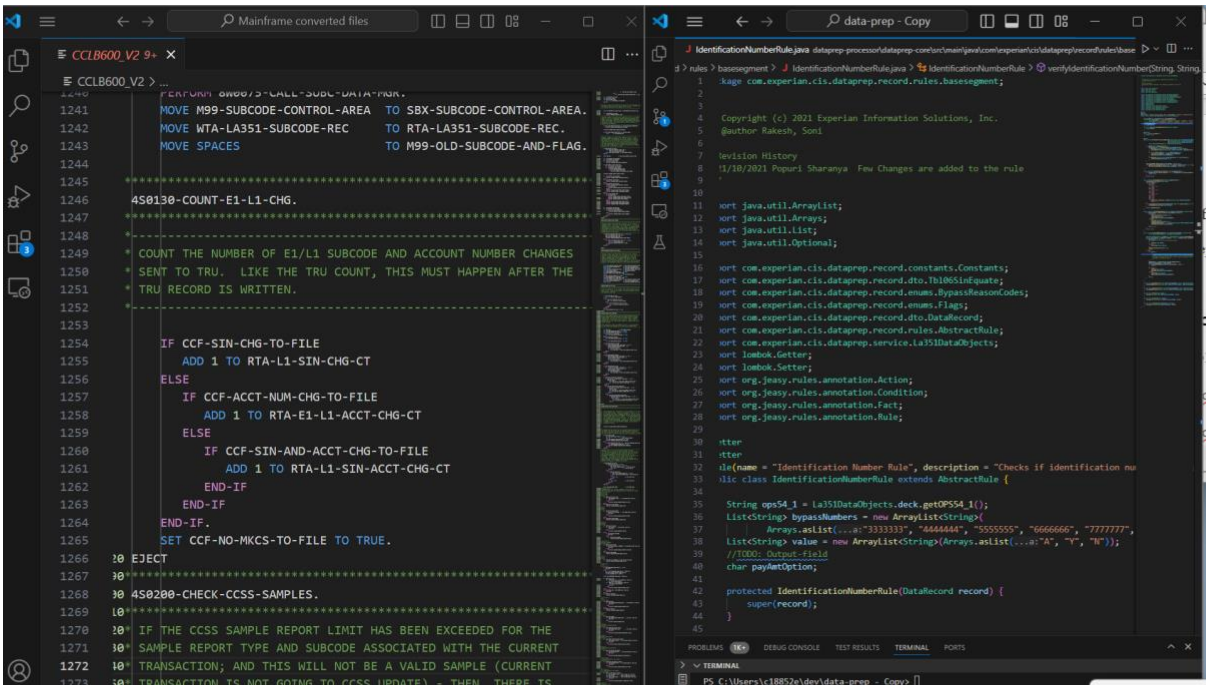
Your objective is to pinpoint where the logic needs to be adjusted. The issue is that the MFfiles repository (written in COBOL) on the left returns 0, while the record repository (written in Java) on the right returns 1. We need to align the logic in the record repository with the MFfiles repository so that both the Java and COBOL code function the same way.

**Note:** Running the program can be a bit challenging, so feel free to message me so I can run it. I would send you the output logs. (I've attached the MFfiles repo, the record repo, and a text file containing the output from my run—no need to worry about messaging me too often; please try to do as much as you can independently.)

# Tracing help

This was my tracing so far. Rpt002Areport.java-> ReportWriterFactory.java -> IdentificationNumberRule.java.

And in our Cobol these are files that represent that logic
`Open CCLB600_V2` and navigate to the `4S0130-COUNT-E1-L1-CHG`


Mainframe converted files
CCLB600_V2 9+
CCLB600_V2 > ...
data-prep - Copy
IdentificationNumberRule.java dataprep-processor\dataprep-core\src\main\java\com\experian\cis\dataprep\record\rules\base
d > rules > basesegment > IdentificationNumberRule.java > IdentificationNumberRule > verifyIdentificationNumber(String, String,
PROBLEMS 1K+
DEBUG CONSOLE
TEST RESULTS
TERMINAL
PORTS
TERMINAL

```
               PERFORM 8W0075-CALL-SUBC-DATA-MGR.
               MOVE M99-SUBCODE-CONTROL-AREA  TO SBX-SUBCODE-CONTROL-AREA.
               MOVE WTA-LA351-SUBCODE-REC     TO RTA-LA351-SUBCODE-REC.
               MOVE SPACES                    TO M99-OLD-SUBCODE-AND-FLAG.

      ******************************************************************
       4S0130-COUNT-E1-L1-CHG.
      ******************************************************************
      *-----------------------------------------------------------------
      * COUNT THE NUMBER OF E1/L1 SUBCODE AND ACCOUNT NUMBER CHANGES
      * SENT TO TRU.  LIKE THE TRU COUNT, THIS MUST HAPPEN AFTER THE
      * TRU RECORD IS WRITTEN.
      *-----------------------------------------------------------------

               IF CCF-SIN-CHG-TO-FILE
                  ADD 1 TO RTA-L1-SIN-CHG-CT
               ELSE
                  IF CCF-ACCT-NUM-CHG-TO-FILE
                     ADD 1 TO RTA-E1-L1-ACCT-CHG-CT
                  ELSE
                     IF CCF-SIN-AND-ACCT-CHG-TO-FILE
                        ADD 1 TO RTA-L1-SIN-ACCT-CHG-CT
                     END-IF
                  END-IF
               END-IF.
               SET CCF-NO-MKCS-TO-FILE TO TRUE.
    20 EJECT
    90******************************************************************
    90 4S0200-CHECK-CCSS-SAMPLES.
    10******************************************************************
    20* IF THE CCSS SAMPLE REPORT LIMIT HAS BEEN EXCEEDED FOR THE
    30* SAMPLE REPORT TYPE AND SUBCODE ASSOCIATED WITH THE CURRENT
    40* TRANSACTION; AND THIS WILL NOT BE A VALID SAMPLE (CURRENT
    50* TRANSACTION IS NOT GOING TO CCSS UPDATE) - THEN, THERE IS
```

```
:kage com.experian.cis.dataprep.record.rules.basesegment;


 Copyright (c) 2021 Experian Information Solutions, Inc.
 @author Rakesh, Soni

:evision History
:1/10/2021 Popuri Sharanya  Few Changes are added to the rule
 '

 ort java.util.ArrayList;
 ort java.util.Arrays;
 ort java.util.List;
 ort java.util.Optional;

 ort com.experian.cis.dataprep.record.constants.Constants;
 ort com.experian.cis.dataprep.record.dto.Tbl06SinEquate;
 ort com.experian.cis.dataprep.record.enums.BypassReasonCodes;
 ort com.experian.cis.dataprep.record.enums.Flags;
 ort com.experian.cis.dataprep.record.dto.DataRecord;
 ort com.experian.cis.dataprep.record.rules.AbstractRule;
 ort com.experian.cis.dataprep.service.La351DataObjects;
 ort lombok.Getter;
 ort lombok.Setter;
 ort org.jeasy.rules.annotation.Action;
 ort org.jeasy.rules.annotation.Condition;
 ort org.jeasy.rules.annotation.Fact;
 ort org.jeasy.rules.annotation.Rule;

:tter
:tter
:le(name = "Identification Number Rule", description = "Checks if identification nu
 lic class IdentificationNumberRule extends AbstractRule {

  String ops54_1 = La351DataObjects.deck.getOPS54_1();
  List<String> bypassNumbers = new ArrayList<String>(
          Arrays.asList(...a:"3333333", "4444444", "5555555", "6666666", "7777777",
  List<String> value = new ArrayList<String>(Arrays.asList(...a:"A", "Y", "N"));
  //TODO: Output-field
  char payAmtOption;

  protected IdentificationNumberRule(DataRecord record) {
      super(record);
  }

```

```
PS C:\Users\c18852e\dev\data-prep - Copy>
```

Tracing explanation:

## 1. Entry Point: `rpt002Areport.java`

- The line `setReportLine("COLL STAT 62 TO DA MED", " SUBCODE CHANGES", getCountValue(data.get(ReportConstants.STATUS_62_DA_MED)), getCountValue(data.get(ReportConstants.L1_SIN_CHANGE)), buf);` indicates that the report is pulling data related to subcode changes.
- The method `getCountValue(data.get(ReportConstants.L1_SIN_CHANGE))` is specifically retrieving the count of `L1_SIN_CHANGE`.

## 2. **`countL1Changes()`** Method:

- The method `countL1Changes()` is responsible for setting the counts related to `L1_SIN_CHANGE`, `L1_ACCT_CHANGE`, and `L1_SIN_ACCT_CHANGE`.
- The logic within this method increments the counters based on the value of the `changeIndicator`.

## 3. Examine **`changeIndicator`**:

- The `changeIndicator` is extracted from `record.getL1Segment().getChangeIndicator()`, which is a key factor in determining which count is incremented.

## 4. Determine Where **`changeIndicator`** is Set:

- To fully understand the logic, it's crucial to trace how and where `changeIndicator` is set or modified.
- As per your earlier information, this is likely happening in the `IdentificationNumberRule` class, particularly within the `setChangeIndicator()` method.

## 5. **`setChangeIndicator()`** in **`IdentificationNumberRule`**:

- This method maps the `ops54_1` value to a specific `changeIndicator`.
- It’s crucial to verify that this method is functioning as intended, ensuring that the correct `changeIndicator` is being set.

## 6. Verify Interaction Between Methods:

- Ensure that the `setChangeIndicator()` method correctly modifies the `changeIndicator` and that the `countL1Changes()` method then correctly increments the appropriate count based on that `changeIndicator`.

The below is my failed recovery attempt:

The logic for incrementing a subcode change is primarily managed by the `countL1Changes()` method in conjunction with the `setChangeIndicator()` method from `IdentificationNumberRule`. You should focus on:

1 Ensuring that `setChangeIndicator()` sets the correct values for `changeIndicator`.
2 Verifying that `countL1Changes()` correctly increments the counts based on `changeIndicator`.
3 Checking for any edge cases where `record.getL1Segment()` could be null or `changeIndicator` could have an unexpected value.

To verify and update the `IdentificationNumberRule` class, we need to ensure that the `changeIndicator` is set correctly based on the `ops54_1` value. I will add some logging and validation to ensure that `changeIndicator` is being set as expected. Here's an updated version of your class with these changes:

```
package com.experian.cis.dataprep.record.rules.basesegment;

import java.util.ArrayList;
import java.util.Arrays;
import java.util.List;
import java.util.Optional;

import com.experian.cis.dataprep.record.constants.Constants;
import com.experian.cis.dataprep.record.dto.Tbl06SinEquate;
import com.experian.cis.dataprep.record.enums.BypassReasonCodes;
import com.experian.cis.dataprep.record.enums.Flags;
import com.experian.cis.dataprep.record.dto.DataRecord;
import com.experian.cis.dataprep.record.rules.AbstractRule;
import com.experian.cis.dataprep.service.La351DataObjects;
import lombok.Getter;
import lombok.Setter;
import lombok.extern.slf4j.Slf4j;
import org.jeasy.rules.annotation.Action;
import org.jeasy.rules.annotation.Condition;
import org.jeasy.rules.annotation.Fact;
import org.jeasy.rules.annotation.Rule;

@Getter
@Setter
@Slf4j
@Rule(name = "Identification Number Rule", description = "Checks if identification number
is valid and set appropriate flag", priority = 1)
public class IdentificationNumberRule extends AbstractRule {

    String ops54_1 = La351DataObjects.deck.getOPS54_1();
    List<String> bypassNumbers = new ArrayList<>(
            Arrays.asList("3333333", "4444444", "5555555", "6666666", "7777777", "8888888",
"9999999"));
    List<String> value = new ArrayList<>(Arrays.asList("A", "Y", "N"));
    char payAmtOption;

    protected IdentificationNumberRule(DataRecord record) {
        super(record);
    }

    @Condition
    public boolean validate() {
        return true;
    }

    @Action
    public void
verifyIdentificationNumber(@Fact(Constants.Keys.IDENTIFICATION_NUMBER) String
identificationNumber,
                                          @Fact(Constants.Keys.L1_IDENTIFICATION_NUMBER)
String newIdentificationNumber,
                                          @Fact(Constants.Keys.CHANGE_INDICATOR) String
changeIndicator)
            throws NoSuchFieldException, IllegalAccessException {

        if (getRecord().l1SegmentExists()) {
            changeIndicator = setChangeIndicator(changeIndicator);
            if (getFlag(Constants.Keys.CHANGE_INDICATOR) == Flags.VALID) {
                setL1SegmentField(Constants.Keys.CHANGE_INDICATOR, changeIndicator);
            } else {
                log.warn("Change indicator is not valid for L1 segment. Value: {}",
changeIndicator);
```

```
            }
            if ("10".equals(ops54_1) || "11".equals(ops54_1)) {
                identificationNumber = newIdentificationNumber;
                updateRecord(identificationNumber);
            }
            if (getFlag(Constants.Keys.CHANGE_INDICATOR) == Flags.INVALID) {
                setDiscardCondition(true);
                log.info("Record discarded due to invalid change indicator.");
            }
        }

        searchCCLATB06_Update(identificationNumber, false);
        if (getRecord().l1SegmentExists() && ("2".equals(changeIndicator) ||
"3".equals(changeIndicator))) {
            searchCCLATB06_Update(newIdentificationNumber, true);
        }
    }

    public boolean isChangeIndicatorValid(String changeIndicator) {
        return "1".equals(changeIndicator) || "2".equals(changeIndicator) ||
"3".equals(changeIndicator);
    }

    public String setChangeIndicator(String changeIndicator) {
        log.debug("Setting change indicator. Current value: {}, ops54_1: {}", changeIndicator,
ops54_1);
        setChangeIndicatorFlag(Flags.VALID);

        switch (ops54_1) {
            case "03":
            case "04":
                return "1";
            case "05":
            case "06":
                return "2";
            case "07":
            case "08":
                return "3";
            case "09":
            case "10":
            case "11":
            case "12":
                getRecord().getFlags().remove(Constants.Keys.CHANGE_INDICATOR);
                return " ";
            default:
                log.error("Unexpected ops54_1 value: {}", ops54_1);
                setChangeIndicatorFlag(Flags.INVALID);
                return changeIndicator;
        }
    }
    public boolean checkAndUpdateBypass(String identificationNumber, boolean isL1)
throws NoSuchFieldException, IllegalAccessException {
        if (bypassNumbers.contains(identificationNumber.trim())) {
            if (isL1) {
                setDiscardCondition(true);
                setL1Flag(Flags.B);
                updateL1Record(identificationNumber);
            } else {
                setFlag(Flags.B);
                setBypassCondition(true);
                setBypassReasonCode("" + -1,
BypassReasonCodes.BYPASS_REQUEST_SUBCODE);
                updateRecord(identificationNumber);
            }
            return true;
        }
        return false;
    }

    public void searchCCLATB06_Update(String identificationNumber, boolean isL1)
            throws NoSuchFieldException, IllegalAccessException {

        Optional<Tbl06SinEquate> sinEquate =
La351DataObjects.service.getTbl06SinEquate(identificationNumber);
        String subscriberId = sinEquate.isEmpty() ? null : sinEquate.get().getSubscriberid();

        if (subscriberId == null) {
            if (!isL1) {
                setFlag(Flags.ERROR);
            } else {
                setL1Flag(Flags.INVALID);
            }
            setRecycleCondition(true);
            return;
        } else {
            payAmtOption = sinEquate.get().getLstpaydtprcind();
            identificationNumber = subscriberId;

            if (checkAndUpdateBypass(identificationNumber, isL1)) {
                return;
            }

            if (!isL1) {
                setFlag(Flags.VALID);
                updateRecord(identificationNumber);
            } else {
                setL1Flag(Flags.VALID);
                updateL1Record(identificationNumber);
            }
        }
    }

    private void updateRecord(String identificationNumber) throws NoSuchFieldException,
IllegalAccessException {
        setBaseSegmentField(Constants.Keys.IDENTIFICATION_NUMBER,
identificationNumber);
    }

    private void updateL1Record(String identificationNumber) throws NoSuchFieldException,
IllegalAccessException {
        setL1SegmentField(Constants.Keys.L1_IDENTIFICATION_NUMBER,
identificationNumber);
    }

    private void setFlag(Flags value) {
        setFlag(Constants.Keys.IDENTIFICATION_NUMBER, value);
    }

    private void setL1Flag(Flags value) {
        setFlag(Constants.Keys.L1_IDENTIFICATION_NUMBER, value);
    }

    private void setChangeIndicatorFlag(Flags value) {
        setFlag(Constants.Keys.CHANGE_INDICATOR, value);
    }
}
```

To verify and update the `IdentificationNumberRule` class, we need to ensure that the `changeIndicator` is set correctly based on the `ops54_1` value. I will add some logging and validation to ensure that `changeIndicator` is being set as expected. Here's an updated version of your class with these changes:

### Updated `IdentificationNumberRule` Class

```
java
package
com.experian.cis.dataprep.record.rules.basesegment;

import java.util.ArrayList;
import java.util.Arrays;
import java.util.List;
import java.util.Optional;

import
com.experian.cis.dataprep.record.constants.Constants;
import
com.experian.cis.dataprep.record.dto.Tbl06SinEquate;
```

```
import
com.experian.cis.dataprep.record.enums.BypassReasonCodes;
import com.experian.cis.dataprep.record.enums.Flags;
import com.experian.cis.dataprep.record.dto.DataRecord;
import
com.experian.cis.dataprep.record.rules.AbstractRule;
import
com.experian.cis.dataprep.service.La351DataObjects;
import lombok.Getter;
import lombok.Setter;
import lombok.extern.slf4j.Slf4j;
import org.jeasy.rules.annotation.Action;
import org.jeasy.rules.annotation.Condition;
import org.jeasy.rules.annotation.Fact;
import org.jeasy.rules.annotation.Rule;

@Getter
@Setter
@Slf4j
@Rule(name = "Identification Number Rule", description =
"Checks if identification number is valid and set
appropriate flag", priority = 1)
public class IdentificationNumberRule extends
AbstractRule {

    String ops54_1 = La351DataObjects.deck.getOPS54_1();
    List<String> bypassNumbers = new ArrayList<>(
            Arrays.asList("3333333", "4444444",
"5555555", "6666666", "7777777", "8888888", "9999999"));
    List<String> value = new
ArrayList<>(Arrays.asList("A", "Y", "N"));
    char payAmtOption;

    protected IdentificationNumberRule(DataRecord record)
{
        super(record);
    }

    @Condition
    public boolean validate() {
        return true;
    }

    @Action
    public void
verifyIdentificationNumber(@Fact(Constants.Keys.IDENTIFIC
ATION_NUMBER) String identificationNumber,
@Fact(Constants.Keys.L1_IDENTIFICATION_NUMBER) String
newIdentificationNumber,

@Fact(Constants.Keys.CHANGE_INDICATOR) String
changeIndicator)
            throws NoSuchFieldException,
IllegalAccessException {

        if (getRecord().l1SegmentExists()) {
            changeIndicator =
setChangeIndicator(changeIndicator);
            if (getFlag(Constants.Keys.CHANGE_INDICATOR)
== Flags.VALID) {

setL1SegmentField(Constants.Keys.CHANGE_INDICATOR,
changeIndicator);
            } else {
                log.warn("Change indicator is not valid
for L1 segment. Value: {}", changeIndicator);
            }
            if ("10".equals(ops54_1) ||
"11".equals(ops54_1)) {
                identificationNumber =
newIdentificationNumber;
                updateRecord(identificationNumber);
            }
            if (getFlag(Constants.Keys.CHANGE_INDICATOR)
== Flags.INVALID) {
                setDiscardCondition(true);
                log.info("Record discarded due to invalid
change indicator.");
            }
        }

        searchCCLATB06_Update(identificationNumber,
false);
        if (getRecord().l1SegmentExists() &&
("2".equals(changeIndicator) ||
"3".equals(changeIndicator))) {

searchCCLATB06_Update(newIdentificationNumber, true);
        }
    }

    public boolean isChangeIndicatorValid(String
changeIndicator) {
        return "1".equals(changeIndicator) ||
"2".equals(changeIndicator) ||
"3".equals(changeIndicator);
    }

    public String setChangeIndicator(String
changeIndicator) {
        log.debug("Setting change indicator. Current
value: {}, ops54_1: {}", changeIndicator, ops54_1);
        setChangeIndicatorFlag(Flags.VALID);

        switch (ops54_1) {
            case "03":
            case "04":
                return "1";
            case "05":
            case "06":
                return "2";
            case "07":
            case "08":
                return "3";
            case "09":
            case "10":
            case "11":
            case "12":

getRecord().getFlags().remove(Constants.Keys.CHANGE_INDIC
ATOR);
                return " ";
            default:
                log.error("Unexpected ops54_1 value: {}",
ops54_1);
                setChangeIndicatorFlag(Flags.INVALID);
                return changeIndicator;
        }
    }

    public boolean checkAndUpdateBypass(String
identificationNumber, boolean isL1) throws
NoSuchFieldException, IllegalAccessException {
        if
(bypassNumbers.contains(identificationNumber.trim())) {
            if (isL1) {
                setDiscardCondition(true);
                setL1Flag(Flags.B);
                updateL1Record(identificationNumber);
            } else {
                setFlag(Flags.B);
                setBypassCondition(true);
                setBypassReasonCode("" + -1,
BypassReasonCodes.BYPASS_REQUEST_SUBCODE);
                updateRecord(identificationNumber);
            }
            return true;
        }
        return false;
    }

    public void searchCCLATB06_Update(String
identificationNumber, boolean isL1)
            throws NoSuchFieldException,
IllegalAccessException {

        Optional<Tbl06SinEquate> sinEquate =
La351DataObjects.service.getTbl06SinEquate(identification
Number);
        String subscriberId = sinEquate.isEmpty() ? null
: sinEquate.get().getSubscriberid();

        if (subscriberId == null) {
            if (!isL1) {
                setFlag(Flags.ERROR);
            } else {
                setL1Flag(Flags.INVALID);
            }
            setRecycleCondition(true);
            return;
        } else {
            payAmtOption =
sinEquate.get().getLstpaydtprcind();
            identificationNumber = subscriberId;

            if
(checkAndUpdateBypass(identificationNumber, isL1)) {
                return;
            }

            if (!isL1) {
                setFlag(Flags.VALID);
                updateRecord(identificationNumber);
            } else {
                setL1Flag(Flags.VALID);
                updateL1Record(identificationNumber);
            }
        }
    }

    private void updateRecord(String
identificationNumber) throws NoSuchFieldException,
IllegalAccessException {

setBaseSegmentField(Constants.Keys.IDENTIFICATION_NUMBER,
identificationNumber);
    }

    private void updateL1Record(String
identificationNumber) throws NoSuchFieldException,
IllegalAccessException {

setL1SegmentField(Constants.Keys.L1_IDENTIFICATION_NUMBER
, identificationNumber);
    }

    private void setFlag(Flags value) {
        setFlag(Constants.Keys.IDENTIFICATION_NUMBER,
value);
    }

    private void setL1Flag(Flags value) {
        setFlag(Constants.Keys.L1_IDENTIFICATION_NUMBER,
value);
    }

    private void setChangeIndicatorFlag(Flags value) {
        setFlag(Constants.Keys.CHANGE_INDICATOR, value);
    }
}
```

## Key Changes and Additions:

1 **Logging:**

- Added `log.debug` and `log.warn` statements to trace the setting of the `changeIndicator` and to monitor the flow within the `setChangeIndicator` and `verifyIdentificationNumber` methods.

2 **Validation and Error Handling:**

- Enhanced validation of the `changeIndicator` to ensure it falls within the expected values (`1`, `2`, or `3`).
- Added a warning log if the `changeIndicator` is not valid and an error log if an unexpected `ops54_1` value is encountered.

The method `countL1Changes()` is widely used in the `ReportWriterFactory` class

1. **Method Logic:**
   - The `countL1Changes()` method starts by checking if `record.getL1Segment()` is `null`. If it is, the method returns without doing anything.
   - It then retrieves the `changeIndicator` from the `L1Segment` and converts it to an integer.
   - Based on the value of `changeIndicator`, it increments different counters:
     - If `changeIndicator` is `2`, it increments the `L1_SIN_CHANGE` counter.
     - If `changeIndicator` is `1`, it increments the `L1_ACCT_CHANGE` counter.
     - If `changeIndicator` is `3`, it increments the `L1_SIN_ACCT_CHANGE` counter.
2. **Assembly Code on the Right:**
   - The right side of the screenshot appears to be a representation of some assembly or low-level language code. This code shows logic similar to what the `countL1Changes()` method is doing, albeit in a different language or format. Specifically, it seems to be checking for conditions and incrementing some counts based on those conditions, just like the method in your Java code.

## Checking Against the Classes:

1. **`L1Segment` Class:**
   - The `L1Segment` class you shared contains a `String changeIndicator`. This is where the `changeIndicator` value originates. It is crucial that this value is correctly set before the `countL1Changes()` method is invoked.
2. **`countL1Changes()` Method:**
   - The method depends entirely on the `changeIndicator` value being accurate. If there is any issue with how `changeIndicator` is being set or retrieved, it would affect the logic in this method.
3. **`IdentificationNumberRule` Class:**
   - This class appears to manipulate or set the `changeIndicator`. Specifically, it has logic to set the `changeIndicator` based on certain conditions, and it checks whether this value is valid.
   - You should ensure that this logic is functioning correctly and that the correct `changeIndicator` values are being set in the `L1Segment`.