User Guide for Java framework for Cobol migration

Ispirer Systems LTD

Minsk 2018

Contents

INTRODUCTION

The framework contains Java classes and interfaces which emulate the behavior of Cobol. This document describes how to work this framework.

Specifically, the framework emulates Cobol variable declarations, as well as reading the values from the variables and updating variable values. It emulates how Cobol reads and writes files. It emulates Cobol's interaction with databases.

The framework emulates many aspects of Cobol. Feel free to use as much or as little of these features as you want.

WORKING WITH VARIABLES AND STRUCTURES

PictureType class was created to imitate Cobol variables so it can read in data from a file or write data to a file in the correct format. PictureType is genericized (similar to List in Java).

If you do not need to use COBOL formats to write formatted data to a file you do not have to use PictureType. You can use usual Java types like String, Integer, Long, BigDecimal…

Declaration and initialization of variables

To declare PictureType variable, you need to determine the type of variable and format. Unlike Java, Cobol variables require you to specify the number of characters or digit. This is called the *format* for the variable. Numeric and string variables use this number for formatting when the value is printed to a file.

There are four types of variables that are supported in the PictureType class:

- `PictureType<Integer>` - need to use for Integer variables
- `PictureType<BigDecimal>` - need to use for BigDecimal variables
- `PictureType<Long>` - need to use for Long variables
- `PictureType<String>` - need to use for String variables

We will abbreviate PictureType as PT, for convenience.

To declare PT variable need to use one of the next patterns

Pattern 1

```
public PictureType<Integer> x = new PictureType<>(PictureType.Type.Integer,
new DecimalFormat("9"));
```

In this pattern, the first argument in the constructor is a PictureType type. The second argument is the format (in this case, a single unsigned digit).

Pattern 2

```
public PictureType<Integer> y = new PictureType<>(new DecimalFormat("9(7)"),
0);
```

In this pattern, the first argument in the constructor is a format, and the second argument is the default value.

Pattern 3

```
public PictureType<String> z = new PictureType<>( PictureType.Type.String,
new AlphanumericFormat("X(2)"), PicutreType.DefaultValue.Spaces);
```

In this pattern, the first argument in the constructor is a PictureType type. The second argument is the format. The third argument is the default value. In this case, the default value is blank spaces.

This is a list of the various PictureType types

Use `PictureType.Type.Integer` if you create `PictureType<Integer>`

Use `PictureType.Type.Long` if you create `PictureType<Long>`

Use `PictureType.Type.BigDecimal` if you create `PictureType<BigDecimal>`

Use `PictureType.Type.String` if you create `PictureType<String>`

Format

There are two types of formats **AlphanumericFormat** and **DecimalFormat** for String variables and variables of numeric types (such as Integer, Long, BigDecimal).

Example of creating AlphanumericFormat:

```
new AlphanumericFormat("X(2)")
```

This indicates the string contains 2 characters

Example of creating DecimalFormat:

```
new DecimalFormat("S9(9)")
```

This indicates a signed number (S means signed) which contains 9 digits.

defaultValue – use any literal or numeric constant to specify default value.

default value that is in (Spaces, Zeroes, HighValues, LowValues…) - use one of the default values: Spaces, Zeroes, HighValues, LowValues…

Cobol uses names like Spaces, Zeroes, HighValues, LowValues as constants that can be used to initialize variables.

Using PictureType<Integer>

This type is using for Integer variable. It means that this variable is not fractional and has length that is less than 10 digits.

Examples of formats:

`S9(9)` – signed numeric format that contains of 9 numeric symbols

`9(9)` – unsigned numeric format that contains of 9 numeric symbols.

`999` – unsigned numeric format that contains of three numeric symbols.

The number in parentheses indicate how many digits there are. If there are no parentheses, such as 999, then the number of digits is based on how often 9 appears. Therefore, 999 means 3 digits.

To learn more about numeric formats read COBOL documentation about Picture clause.

Using PictureType<Long>

This type is using for `Long` variable. It means that this variable is not fractional and has length that is greater than or equal to 10 digits.

Examples of formats:

`S9(10)` – signed numeric format that contains of 10 numeric symbols

`9(15)` –unsigned numeric format that contains of 15 numeric symbols.

To learn more about numeric formats read COBOL documentation about Picture clause.

Using PictureType<BigDecimal>

This type is using for BigDecimal variable. It means that this variable is fractional.

Examples of formats:

`S9(9)V99` – signed numeric format that contains of 11 numeric symbols, 2 of which are fractional part

`9(9)V9(3)` –unsigned numeric format that contains of 12 numeric symbols, 3 of which are fractional part.

`999V999999` – unsigned numeric format that contains of 9 numeric symbols, 6 of which are fractional part.

To learn more about numeric formats read Cobol documentation about Picture clause.

Using PictureType<String>

This type is using for String variable.

Examples of formats:

`X(25)` – string variable that contains of 25 characters.

To learn more about alphanumeric formats read Cobol documentation about Picture clause.

Using of variables

Set/Get value

To set value into PT variable, use `setValue(new Value)`.

To get value from PT variable, use `getValue()`.

Comparison

To compare PT variable with any value or other variable using relational operators (equality, greater than, less than, etc), use the `compareTo` method:

```
Var1.compareTo(Var2)
```

This method returns one of three values:

- `1` if `Var1` is greater than `Var2`
- `0` if `Var1` is equal to `Var2`
- `-1` if `Var1` is less than `Var2`

Unlike Cobol, comparison operators like > or < can't be used in Java to compare objects (which is why this method is needed) since Java does not support operator overloading.

Math operations

You can perform mathematic operations as addition, subtraction, multiplication, division with `PictureType` on numeric types, e.g. `BigDecimal`.

ADDITION

To perform addition you can use one of the next methods:

```
public PictureType<BigDecimal> add(Object addValue)
```

This method adds `addValue` to this variable without rounding.

```
public PictureType<BigDecimal> addTrunc(PictureType finalObject, Object
      addValue)
```

This method adds `addValue` to this variable and rounds the result.

Use this method if you need to round each small calculation separately.

SUBTRACTION

To perform subtraction you can use one of the next methods:

```
public PictureType<BigDecimal> subtract(Object addValue)
```

This method subtracts addValue from variable without rounding.

```
public PictureType<BigDecimal> subtractTrunc(PictureType finalObject, Object
      addValue)
```

This method subtracts `addValue` from variable and rounds the result. Use this method if you need to round each small calculation separately.

MULTIPLICATION

To perform multiplication you can use one of the next methods:

```
public PictureType<BigDecimal> multiply(Object addValue)
```

This method multiplies `addValue` by variable without rounding.

```
public PictureType<BigDecimal> multiplyTrunc(PictureType finalObject, Object
      addValue)
```

This method multiplies `addValue` by variable and rounds the result. Use this method if you need to round each small calculation separately.

DIVISION

To perform division you can use one of the next methods:

```
public PictureType<BigDecimal> divide(Object addValue)
```

This method divides variable by `addValue` without rounding.

```
public PictureType<BigDecimal> divideTrunc(PictureType finalObject, Object
      addValue)
```

This method divides variable by `addValue` and rounds the result. Use this method if you need to round each small calculation separately.

You can also use `divideAndRemainder` method. It returns array of `BigDecimal` variables of size 2. The first element (index 0 in array) is the integer part of division and the second element (index 1 in array) is a remainder of the division.

You can combine these methods and write different calculation expressions.

SETTING THE VALUE OF A VARIABLE

To set the result of calculation into any variable you can use one of methods:

```
setValue(new value)
```

Used to set the result of calculation into result variable without rounding. If you've used Trunc methods (e.g., addTrunc, subtractTrunc, etc.) you can use this method to set value because the result is already rounded.

```
setRoundedValue(new value that should be rounded)
```

Used to set the rounded result of calculation into result variable. Use this method if you need to round the result of whole expression.

Work with Structures

A structure in Cobol is similar to a class in Java, except all fields are public, and there are no methods.

How to create class for Structure

To create class for data structure need to:

1) Create class that extends StructureModel. Example:

```
public class DataStructure extends StructureModel{

}
```

2) Add variable sizes. This array stores sizes of all fields in structure. It is used to fill each field in DS if you set value to whole structure. You need to add next field:

```
private int sizes[];
```

Example of class after this step:

```
public class DataStructure extends StructureModel{
    private int sizes[];
}
```

3) Add fields as public fields. Example:

```
public class DataStructure extends StructureModel{
    private int sizes[];
    PictureType<Integer> intField = new PictureType<>(PictureType.Type.Integer, new
       DecimalFormat("999"));
    PictureType<String> strField = new PictureType<>(PictureType.Type.String, new
       AlphanumericFormat("XXX"));
    PictureType<BigDecimal> bDField = new PictureType<>(PictureType.Type.BigDecimal, new
       DecimalFormat("99V99"));
}
```

4) Add constructor. In constructor, you need to initialize `sizes` array. This array has to be filled by sizes of all public fields in this class. To get size of PT variable or other structure you can use getSize() method.

Example of class after this step:

```
public class DataStructure extends StructureModel{
    private int sizes[];
    PictureType<Integer> intField = new PictureType<>(PictureType.Type.Integer, new
       DecimalFormat("999"));
    PictureType<String> strField = new PictureType<>(PictureType.Type.String, new
       AlphanumericFormat("XXX"));
    PictureType<BigDecimal> bDField = new PictureType<>(PictureType.Type.BigDecimal, new
       DecimalFormat("99V99"));

    public DataStructure() {
        sizes = new int[]{intField.getSize(), strField.getSize(), bDField.getSize()};
    }
}
```

5) Implement methods.

You have to implement following methods :

- public String toString(). This method has to return a string representing the object. String representing of the structure is a concatenation of string representation of all its fields.

Example:

```
@Override
    public String toString() {
        return "" + intField + strField + bDField;
    }
```

`public void setData(char[] data)`
`public void setData(String[] data)`

These methods are used to set data into all fields in the structure. In the `setData(char[] data)`, the data that should be set into structure in array of chars and composes these chars to strings using sizes variable. Finally, it has to get an array of values that will be set into each field and send this array into `setData(String[] data).`

setData for char[] array parameter is usually implemented as follows:

```
@Override
    public void setData(char[] data) {
        setData(getStringValues(data, sizes));
    }
```

In the `setData(String[] data)`, set corresponding element of array into each field in the class.
For Numeric fields use:

```
  numField.setDataFromFile(data[i].getBytes());
```

For String fields, use:

```
          strField.setValue(data[i]);
```

Index of array starts from 0 in java.

Here's a typical implementation for `setData` with a String array as parameter:

```
    @Override
    public void setData(String[] data) {
        intField.setDataFromFile(data[0].getBytes());
        strField.setValue(data[1]);
        bDField.setDataFromFile(data[2].getBytes());
    }
```

`public int getSize().`

This method returns size of structure. Size of structure is a sum of sizes of its fields.
Here's a typical implementation:

```
  @Override
    public int getSize() {
        int sum = 0;
        for(int i = 0; i < sizes.length; i ++) {
            sum += sizes[i];
        }
        return sum;
    }
```

`public void initialize();`

This method should initialize each field in the class.

Here's a typical implementation:

```
    @Override
    public void initialize() {
        intField.initialize();
        strField.initialize();
        bDField.initialize();
    }
```

`public byte[] toFile();`

This method creates a byte array representation of a structure so it can be written to a file. To get a byte array representation of structure, you need to concatenate byte array representations of all fields in structure.

To get a byte array representation of a field, call the method on the field:

`field.toFile()`

Here's a typical implementation of `toFile` for a structure with three fields.

Example:

```
  @Override
    public byte[] toFile() {
        byte[] structure = new byte[] { };
        structure = ArrayUtils.addAll(structure, intField.toFile());
        structure = ArrayUtils.addAll(structure, strField.toFile());
        structure = ArrayUtils.addAll(structure, bDField.toFile());
        return structure;
    }
```

`public void setDataFromFile(byte[] bytes);`

This method reads data in a byte array (which has been read from one record in a data file) and sets the values of the fields of the structure. This needs to be implemented.

The first step is to make sure the byte array has a valid length.

This is done by the method, `getFullArray`, as shown below:

```
bytes = getFullArray(bytes);
```

Then, you extract parts of the byte array and initialize each of the fields within the structure. The following code example illustrates this:

```
@Override
    public void setDataFromFile(byte[] bytes) {
        bytes = getFullArray(bytes);
        intField.setDataFromFile(Arrays.copyOf(bytes, intField.getSize()));
        bytes = Arrays.copyOfRange(bytes, intField.getSize(), bytes.length);
        strField.setDataFromFile(Arrays.copyOf(bytes, strField.getSize()));
        bytes = Arrays.copyOfRange(bytes, strField.getSize(), bytes.length);
        bDField.setDataFromFile(Arrays.copyOf(bytes, bDField.getSize()));
        bytes = Arrays.copyOfRange(bytes, bDField.getSize(), bytes.length);
    }
```

`public void setDefaultValue(PictureType.DefaultValue value);`
This method sets each field in the structure to a default value. For PT variables, use method `setDefaultValueFromStructure`.
Example:

```
    @Override
    public void setDefaultValue(PictureType.DefaultValue value) {
        intField.setDefaultValueFromStructure(value);
        strField.setDefaultValueFromStructure(value);
        bDField.setDefaultValueFromStructure(value);
    }
```

This is a complete example of all the steps put together.

```
public class DataStructure extends StructureModel {
    private int sizes[];
    PictureType<Integer> intField = new PictureType<>(PictureType.Type.Integer, new
       DecimalFormat("999"));
    PictureType<String> strFiled = new PictureType<>(PictureType.Type.String, new
       AlphanumericFormat("XXX"));
    PictureType<BigDecimal> bDField = new PictureType<>(PictureType.Type.BigDecimal, new
       DecimalFormat("99V99"));

    public DataStructure() {
        sizes = new int[]{intField.getSize(), strFiled.getSize(), bDField.getSize()};
    }

    @Override
    public String toString() {
        return "" + intField + strFiled + bDField;
    }

    @Override
    public void setData(char[] data) {
        setData(getStringValues(data, sizes));
    }

    @Override
    public void setData(String[] data) {
        intField.setDataFromFile(data[0].getBytes());
        strFiled.setValue(data[1]);
```

```
        bDField.setDataFromFile(data[2].getBytes());
    }

    @Override
    public int getSize() {
        int sum = 0;
        for(int i = 0; i < sizes.length; i ++) {
            sum += sizes[i];
        }
        return sum;
    }

    @Override
    public void initialize() {
        intField.initialize();
        strFiled.initialize();
        bDField.initialize();
    }

    @Override
    public byte[] toFile() {
        byte[] structure = new byte[] { };
        structure = ArrayUtils.addAll(structure, intField.toFile());
        structure = ArrayUtils.addAll(structure, strFiled.toFile());
        structure = ArrayUtils.addAll(structure, bDField.toFile());
        return structure;
    }

    @Override
    public void setDataFromFile(byte[] bytes) {
        bytes = getFullArray(bytes);
        intField.setDataFromFile(Arrays.copyOf(bytes, intField.getSize()));
        bytes = Arrays.copyOfRange(bytes, intField.getSize(), bytes.length);
        strFiled.setDataFromFile(Arrays.copyOf(bytes, strFiled.getSize()));
        bytes = Arrays.copyOfRange(bytes, strFiled.getSize(), bytes.length);
        bDField.setDataFromFile(Arrays.copyOf(bytes, bDField.getSize()));
        bytes = Arrays.copyOfRange(bytes, bDField.getSize(), bytes.length);
    }

    @Override
    public void setDefaultValue(PictureType.DefaultValue value) {
        intField.setDefaultValueFromStructure(value);
        strFiled.setDefaultValueFromStructure(value);
        bDField.setDefaultValueFromStructure(value);
    }
}
```

How to work with Structure object

Creating of object

How to declare and initialize a structure.

Example:

```
public DataStructure dataStructure = new DataStructure();
```

Set value

`setData` requires a char array to set the value of the data structure.

```
dataStructure.setData("value".toCharArray());
dataStructure.setData(otherStructure.toString().toCharArray());
dataStructure.setData(someField);
```

Get string representation

Example:

```
dataStructure.toString()
```

Comparison

To compare structure object with any other object use the same rules as for PT variables

WORKING WITH FILES

The following classes in the framework are used for handling files analogously to how Cobol handles files:

- **FileDescription**. This framework class is created to interact with a single file. Using this class, you can open files in the different modes. You can also read, write, sort, and update records and other operations.
- **FileComparator**. This class sorts records by sort keys.
- **SortKeys**. This class is a helper for sorting records in files. You should use it to declare all keys for sorting.

To work with file you need:

1. Create a FileDescription object
2. Specify the record type
3. Use FileDescription object and record to implement correct business logic

How to create FileDescription object

This is the constructor for `FileDescription`:

```
public FileDescription(String name, int record, int block, boolean isNotEBCDIC)
```

- `name` – path to file
- `record` – length of one record (in characters)
- `block` – count of records in the one block
- `isNotEBCDIC` – specify file encoding.
    - `true` value means ASCII
    - `false` means EBCDIC.

Example:

```
FileDescription exampleFD = new FileDescription("folder/exampleFile.txt", 100, 1, true);
```

A `FileDescription` object has been associated with a file named `exampleFile.txt` file. The length of record is 100 ASCII characters.

How to create record

In a database, a record is one row of a table. In a data file, a record is one row in the file. A record contains fields related to a single entity.

To create a record for use with files, create a class that extends `StructureModel`.

Types that extend `StructureModel`:

- **Structures**. Classes that describes some Type that contains of more than one filed.
- **IntField**. Specific type for records of Integer type.
- **BigDecimalField**. Specific type for records of BigDecimal type.
- **LongFileld**. Specific type for records of Long type.
- **StringField**. Specific type for records of String type.

It is possible to use structures and single fields as a record. Single fields could be `String`, `Integer`, `Long`, `BigDecimal`.

Structure as record

If you want to use a structure as record, you need:

1. Create class for this structure as described in `DataStructure` or use a previously created structure.
2. Declare object of created class and initialize it.

Example:

```
private StructureTypeForFile structureObject = new StructureTypeForFile();
```

Here we created private object of `StructureTypeForFile` class that we will use as a record

String record

If you want to use `String` record, you need to create StrField object.

There are constructor in the `StrField`:

```
public StrField(String format, StructureModel... objs);
```

where

- `format` - is any AlphanumericFormat string
- `objs` – redefined objects //leave empty if don't use redefine

Example:

```
private StrField stringRecord = new StrField("X(123)");
```

Here we have created a private object with type `StrField` whose length is 123 characters. We will use the variable as a record.

Integer record

If you want to use Integer record, you need to create `IntField` object.

There are constructor in the `IntField`:

```
public IntField (String format, StructureModel... objs)
```

Here

- `format` – is a DecimalFormat string for Integer variables. The count of symbols have to be less than 10.
- `objs` – redefined objects //usually is empty for creating record

Example:

```
private IntField intRecord = new IntField ("9(3)");
```

`intRecord` is a field with type `IntField` with width of 3 digits which can be used as a record

Long record

If you want to use Long record, you need to create `LongField` object.

This is the constructor for `LongField`:

```
public LongField (String format, StructureModel... objs)
```

Here

- `format` – is a DecimalFormat string for Long variables. The count of symbols have to be more than 10 or equal to 10.
- `objs` – redefined objects //usually is empty for creating record

Examples:

```
private LongField longRecord = new LongField ("9(13)");
```

`longRecord` is a field with type `LongField` with width of 13 digits which can be used as a record

BigDecimal record

If you want to use BigDecimal record, you need to create BigDecimalField object.

This is the constructor for `BigDecimalField`:

```
public BigDecimalField (String format, StructureModel... objs)
```

Here:

- `format` – is a DecimalFormat string for BigDecimal variables

- `objs` – redefined objects //usually is empty for creating record

Example:

```
private BigDecimalField bigDecimalRecord = new BigDecimalField("9(13)V9(2)");
```

`bigDecimalRecord` is a field with type `BigDecimalField` with width of 13 digits where 2 digits are after the decimal point which can be used as a record

How to work with files

Using `FileDescription`, you can:

- Open file
- Read record
- Write record
- Sort file
- Close file

Opening file

You can open file for Input, Output. Input mode is for reading from a file. Output mode is for writing to a file.

Open file in Input mode

To open a file in Input mode, use `openInput`:

```
public void openInput(StructureModel status) throws IOException
```

Here:

- `status` – status variable. Use null if do not have status variable.

Don't forget to put this method into try/catch block.

Example:

```
try {
    exampleFD.openInput(null);
} catch(IOException e) {
  //handle errors that was caused when it tried to open file
}
```

Open file in Output mode

To open file in Output mode, use `openOutput`:

```
public void openOutput(StructureModel status, boolean append) throws IOException
```

Here:

- `status` – status variable. Use null if you do not have status variable
- `append` – boolean variable that manage clearing of file.
    - `true` – doesn't clear file (appends to end)
    - `false` – clears file

Don't forget to put this method into try/catch block.

Example:

```
try {
   exampleFD.openOutput(null, false);
} catch(IOException e) {
//handle errors that was caused when it tried to open file
}
```

In this example, the file was opened in output mode, and will clear the file before appending.

Reading file

To read a file, use a FileDescription object where you have called openInput.

FileDescription processes a file by reading it line by line. The read method will read one line from a file and place the data in a record.

```
public String read(StructureModel strRec) throws IOException
```

Here:

- strRec – is a record object that corresponds to this file.

Do not forget to put this method into try/catch block. For read statements, you can use specific try/catch block as you can see in the example below.

Example:

```
try {
         exampleFD.read(exampleRec);
      } catch(IOException e) {
         if(exampleFD.getCodeError() != 0) {
            //handle some errors that was thrown while file was reading
         } else {
            // handle EOF
         }
      }
```

In addition, you can put the result of reading into any variable. If you need to save it not just in the record object.

Writing file

You can write into file record or any String object.

Writing a record to a file

To write a single record to a file, use this `write` method:

```
public void write(StructureModel model) throws IOException
```

Here:

- **models** – record object corresponding to this file. Actually, you can use any other `StructureModel` object

Don't forget to put this method into try/catch block.

Example:

```
try {
    EmailFile.write(EmailRec);
} catch(IOException e) {
  // handle some errors that was thrown while record was writing
}
```

Writing a string to a file

To write a string to a file, use this `write` method:

```
public void write(String data) throws IOException
```

Here:

- data – string to write.

Don't forget to put this method into try/catch block.

Example:

```
try {
   EmailFile.write("some string to write");
} catch(IOException e) {
   // handle some errors that was thrown while record was writing
}
```

Sorting file

Sorting of file is implemented Sort statement from COBOL. It sort data from source file and store it into target file.

To sort file need to have:

- Source file. It is file with data that wants to be sorted.
- Target file. It is file to store result. Target and source files can be the same file
- Sorting structure. Structure that declare position of the keys in the record.

Scheme of sorting:

```
// Creating List of SortKeys
List<SortKeys> keys = new ArrayList<>();
keys.add(0, new SortKeys(SortKeys.Keys.ASCENDING, "FirstKey"));
keys.add(1, new SortKeys(SortKeys.Keys.DESCENDING, "SecondKey"));
// Opening Source file in the Input mode
SourceFile.openInput(null);
// Sorting all records and saving the result into List
List list = SourceFile.sortFile(keys, Sorting structure);
// Closing source file
SourceFile.close();
// Opening Target file in Output Mode
TargetFile.openOutput(null, false);
// Write all record that was sorted from list into target file
TempFile.writeAll(list);
```

Do not forget to put this method into try/catch block.

Closing file

As with most file objects, you should close a `FileDescription` object when you are done reading input or writing output by calling the `close` method

```
public void close() throws IOException
```

Don't forget to put this method into try/catch block.

Example:

```
try {
         exampleFD.close();
      } catch(IOException e) {
// handle some errors that was thrown while file was closing
      }
```

WORK WITH DB

Using this framework, you can do next kind of queries:

- SELECT(note that in COBOL SELECT query returns only one record)
- UPDATE/INSERT/DELETE
- CURSOR(declaration, opening, fetching)

Executing Select queries

This is an example of executing a “select” query that returns one record and storing the data into appropriate PictureType variables:

```
try {

         PreparedStatement st =
TransactionManager.getInstance().getConnection().prepareStatement(write here your
query);

//set parameters if needed

         st.setMaxRows(1); //in Cobol Select query returns only 1 row.

         ResultSet rs = st.executeQuery();//executing of select query

         if(rs != null && rs.next()) {

            //set values from db to variables

            TransactionManager.setSuccess();// handle success execution

         } else {

            TransactionManager.getInstance().handleNotFound(); // handle not found

         }

         st.close();

      } catch(SQLException se) {

         TransactionManager.getInstance().handleError(se); //handle error

      }
```

Once the query is made, you will want to extract the values of fields and save them to a variable whose type is compatible with the field from the record that was fetched from the database.

If variable is PictureType<String> use:

```
variable.setValue(rs.getString(i), rs.getMetaData().getColumnType(i));
```

If variable is PictureType<Integer> use:

```
variable.setValue(rs.getInt(i));
```

If variable is PictureType<Long> use:

```
variable.setValue(rs.getLong(i));
```

If variable is PictureType<BigDecimal> use:

```
variable.setValue(rs.getBigDecimal(i));
```

If variable is String use:

```
variable = rs.getString(i);
```

If variable is Integer use:

```
variable = rs.getInt(i);
```

If variable is Long use:

```
variable = rs.getLong(i);
```

If variable is BigDecimal use:

variable = rs.getBigDecimal(i);

This is an example of a select query which returns more than one record:

```
try {
         PreparedStatement st =
TransactionManager.getInstance().getConnection().prepareStatement(write here your
query);
//set parameters if needed
         ResultSet rs = st.executeQuery();//executing of select query
         while(rs != null && rs.next()) {
            //set values from db to variables
            //proceed data
            TransactionManager.setSuccess();// handle success execution
         } else {
            // handle end of resultSet
         }
         st.close();
      } catch(SQLException se) {
         TransactionManager.getInstance().handleError(se); //handle error
      }
```

You may want to put the data in a Java object, and add it to a list of such objects, so it can be processed later.

Executing Update/Delete/Insert queries

Update/delete/insert query are implemented differently from select. Here is an example.

```
try {

    PreparedStatement st =
TransactionManager.getInstance().getConnection().prepareStatement(write here your
query);

//set parameters if needed

     st.execute();//executing of modify query

     if(st.getUpdateCount() < 1) {

         TransactionManager.getInstance().handleNotFound();//handle situation where no
records to modify

     } else {

          TransactionManager.setSuccess();

     }

     st.close();

 } catch(SQLException se) {

      TransactionManager.getInstance().handleError(se); //handle error

 }
```

Work with Cursors

A cursor is basically an iterator for a select query that processes records retrieved from a database, one record at a time. PL/SQL controls the context area through a cursor. A cursor holds the rows (one or more) returned by a SQL statement. The set of rows the cursor holds is referred to as the active set.

Explicit cursors are programmer-defined cursors for gaining more control over the context area. It is created on a SELECT statement that returns more than one row.

Working with an explicit cursor includes the following steps –

- Declaring the cursor for initializing the memory
- Opening the cursor for allocating the memory
- Fetching the cursor for retrieving the data
- Closing the cursor to release the allocated memory

We implemented cursors in the framework because explicit cursors are not supported in java.

To declare cursor need to use:

```
TransactionManager.getInstance().declareCursor(name of cursor, select query for
cursor);
```

To open a cursor:

```
// set parameters before query if needed

TransactionManager.getInstance().getCursors().get(name of cursor).open();
```

To fetch a cursor:

```
try {

   ResultSet rs = TransactionManager.getInstance().getCursors().get(name of
cursor).getResultSet();

   if(rs != null && rs.next()) {

       //set values from db to variables

       //to set values need to use rules that was described for executing Select
queries

        TransactionManager.setSuccess();

    } else {

        TransactionManager.getInstance().handleNotFound(); //handle end of cursor

    }

 } catch(SQLException se) {

    TransactionManager.getInstance().handleError(se);//handle error

 }
```

Once you have processed all the records you want, you need to close cursor, as follows:

```
TransactionManager.getInstance().getCursors().get(name of cursor).close();
```

NOTE: The query executes when cursor opens.

If you need to set parameters in the cursor query, you need to set the parameters prior to opening the cursor, but after the cursor's declaration.

Use:

```
TransactionManager.getInstance().getCursors().get(name of cursor).getStatement().setObject(i,
object);
```

Choose correct method for parameter in the next article.

Queries with parameters

If you need to execute query with parameters need to do next steps:

- Need to put “?”s in the query. The “?” in the SQL query is a placeholder for parameters to go in.
- Need to set parameters values for query before executing.

How to set parameter to query correct? It depends from type of variable.

If variable is PictureType<String> use:

```
st.setObject(i, variable.getTrimmedValue());
```

If variable is PictureType<Integer> use:

```
st.setInt(i, variable.getValue());
```

If variable is PictureType<Long> use:

```
st.setLong(i, variable. getValue ());
```

If variable is PictureType<BigDecimal> use:

```
st.setBigDecimal(i, variable. getValue ());
```

If variable is String use:

```
st.setString(i, variable);
```

If variable is Integer use:

```
st.setInt(i, variable);
```

If variable is Long use:

```
st.setLong(i, variable);
```

If variable is BigDecimal use:

```
st.setBigDecimal(i, variable);
```

Commit and rollback

To commit use:

```
TransactionManager.getInstance().commit();
```

To rollback use:

```
TransactionManager.getInstance().rollback();
```

Executing other queries

If you need to execute some other kind of query, we suggest you read more about JDBC and learn how to implement it in JDBC. Our framework is based on JDBC principles and should handle it.

To get connection object use:

```
TransactionManager.getInstance().getConnection()
```

In addition, you can write your implementation of SELECT, UPDATE, INSERT, DELETE queries. Feel free to choose what is better for you in developing.

USE CASES

Use Case 1: Developers add a new field to the database. How to implement a support for that field in the application?

For example there are table Employee in the DB. It has next fields:

- EMPNO : NUMBER(11,0) – identifier number of Employee
- ENAME : VARCHAR2(15) – name of Employee
- ESALARY : NUMBER(9,2) – salary of Employee
- ESTID : NUMBER(3,0) – establishment id

You can create model class for this structure, or you can create variables separate.

Here you will find an implementation with model class for this table

The most important thing in this example is to choose correct types of variables.

When you have NUMBER type in DB it means that it is one of numeric types in java(PictureType<Integert>, PictureType<BigDecimal>, PictureType<Long>).

Here NUMBER(11,0) will be converted as PictureType<Long> variable with format 9(11)

Here NUMBER(9,2) will be converted as PictureType<BigDecimal> variable with format 9(9)V9(2)

Here NUMBER(3,0) will be converted as PictureType<Integert> variable with format 9(3)

When you have VARCHAR2 type in DB it means that it is PictureType<String> in java

Here VARCHAR2(15) will be converted as PictureType<String> variable with format x(15)

When you finish all steps from explanation how to create structures, you will get next class:

```
public class Employee extends StructureModel {

    private int[] sizes;

    PictureType<Long> empno = new PictureType<>(PictureType.Type.Long, new
DecimalFormat("9(11)"));

    PictureType<String> ename = new PictureType<>(PictureType.Type.String, new
AlphanumericFormat("x(15)"));

    PictureType<BigDecimal> esalary = new PictureType<>(PictureType.Type.BigDecimal,
new DecimalFormat("9(9)v9(2)"));

    PictureType<Integer> estid = new PictureType<>(PictureType.Type.Integer, new
DecimalFormat("9(3)"));


    public Employee() {

        sizes = new int[]{empno.getSize(),ename.getSize(),
esalary.getSize(),estid.getSize()};
```

```
}

@Override
public String toString() {
    return "" + empno + ename + esalary + estid;
}

@Override
public void setData(char[] data) {
    setData(getStringValues(data, sizes));
}


@Override
public void setData(String[] data) {
    empno.setDataFromFile(data[0].getBytes());
    ename.setValue(data[1]);
    esalary.setDataFromFile(data[2].getBytes());
    estid.setDataFromFile(data[3].getBytes());
}

@Override
public int getSize() {
    int sum = 0;
    for(int i = 0; i < sizes.length; i ++) {
        sum += sizes[i];
    }
    return sum;
}

@Override
public void initialize() {
    empno.initialize();
    ename.initialize();
```

```
    esalary.initialize();
    estid.initialize();
}

@Override
public byte[] toFile() {
    byte[] structure = new byte[] { };
    structure = ArrayUtils.addAll(structure, empno.toFile());
    structure = ArrayUtils.addAll(structure, ename.toFile());
    structure = ArrayUtils.addAll(structure, esalary.toFile());
    structure = ArrayUtils.addAll(structure, estid.toFile());
    return structure;
}

@Override
public void setDataFromFile(byte[] bytes) {
    bytes = getFullArray(bytes);
    empno.setDataFromFile(Arrays.copyOf(bytes, empno.getSize()));
    bytes = Arrays.copyOfRange(bytes, empno.getSize(), bytes.length);
    ename.setDataFromFile(Arrays.copyOf(bytes, ename.getSize()));
    bytes = Arrays.copyOfRange(bytes, ename.getSize(), bytes.length);
    esalary.setDataFromFile(Arrays.copyOf(bytes, esalary.getSize()));
    bytes = Arrays.copyOfRange(bytes, esalary.getSize(), bytes.length);
    estid.setDataFromFile(Arrays.copyOf(bytes, estid.getSize()));
    bytes = Arrays.copyOfRange(bytes, estid.getSize(), bytes.length);
}

@Override
public void setDefaultValue(PictureType.DefaultValue value) {
    empno.setDefaultValueFromStructure(value);
    ename.setDefaultValueFromStructure(value);
    esalary.setDefaultValueFromStructure(value);
    estid.setDefaultValueFromStructure(value);
}
```

```
}
```

Then you can create object of this class and use it in program.

Examples of using this class of fields in queries:

Select one record from DB:

```
  public void selectEmployee(){

        try {

            PreparedStatement st =
TransactionManager.getInstance().getConnection().prepareStatement("SELECT EMPNO,
ENAME, ESALARY, ESTID " +

                    " FROM Employee");

            st.setMaxRows(1); //selects only one row

            ResultSet rs = st.executeQuery(); //executes query

            if(rs != null && rs.next()) {

                employee.empno.setValue(rs.getLong(1));

                employee.ename.setValue(rs.getString(2),
rs.getMetaData().getColumnType(2));

                employee.esalary.setValue(rs.getBigDecimal(3));

                employee.estid.setValue(rs.getInt(4));

                TransactionManager.setSuccess();

            } else {

                TransactionManager.getInstance().handleNotFound();

            }

            st.close();

        } catch(SQLException se) {

            TransactionManager.getInstance().handleError(se);

        }

        //print record that was read

        LOGGER.info("identifier number of Employee - " + employee.empno);

        LOGGER.info("name of Employee - " + employee.ename);

        LOGGER.info("salary of Employee - " + employee.esalary);

        LOGGER.info("establishment id - " + employee.estid);

    }
```

To get more info about executing select queries and using PT variables in it read Executing Select queries.

Insert new employee into DB:

```
  public void insertEmployee(){

        try {

            PreparedStatement st =
TransactionManager.getInstance().getConnection().prepareStatement("INSERT INTO
Employee" +

                    " (" +

                    " EMPNO ," +

                    " ENAME ," +

                    " ESALARY ," +

                    " ESTID " +

                    " )" +

                    " VALUES" +

                    " (" +

                    " ?," +

                    " ?," +

                    " ?," +

                    " ?" +

                    " )");

            st.setObject(1, employee.empno.getValue());

            st.setObject(2, employee.ename.getTrimmedValue());

            st.setObject(3, employee.esalary.getValue());

            st.setObject(4, employee.estid.getValue());

            st.execute();

            if(st.getUpdateCount() < 1) {

                TransactionManager.getInstance().handleNotFound();

            } else {

                TransactionManager.setSuccess();

            }

            st.close();

        } catch(SQLException se) {

            TransactionManager.getInstance().handleError(se);
```

```
        }
    }
```

To get more info about Insert queries read Executing Update/Delete/Insert queries.

To get more info about using parameters in the queries read Queries with parameters.

Use Case 2. There is loop for processing some DB records. For each record within the loop developers have to make another DB call and send the ID or a current record as a parameter.

For example, we also have table Establishment. It has next fields:

- ESTID : NUMBER(3,0) – Id of establishment
- ESTNAME : VARCHAR(15) – name of establishment

We need to print names of establishment where all employers work.

First step is to get employers. Then we should process all employers in a loop and find establishment's name for each employee.

You can see the code that does it:

```
  public void selectEstNameForEmployers(){

        try {

            PreparedStatement st =
TransactionManager.getInstance().getConnection().prepareStatement("SELECT EMPNO,
ENAME, ESALARY, ESTID " +

                    " FROM Employee");

            //selects all rows

            ResultSet rs = st.executeQuery(); //executes query

            while(rs != null && rs.next()) { // need to use loop to process all rows

                employee.empno.setValue(rs.getLong(1)); //fill one record

                employee.ename.setValue(rs.getString(2),
rs.getMetaData().getColumnType(2));

                employee.esalary.setValue(rs.getBigDecimal(3));

                employee.estid.setValue(rs.getInt(4));

                TransactionManager.setSuccess();

       // search establishment's name by ESTID value for each employee

                   PreparedStatement st2 =
TransactionManager.getInstance().getConnection().prepareStatement("SELECT ESTNAME " +

                            " FROM Establishment " +

                            " WHERE ESTID = ?");

                    st2.setInt(1,employee.estid.getValue());

                    st2.setMaxRows(1); //selects only one row

                    ResultSet rs2 = st.executeQuery(); //executes query

                    if(rs2 != null && rs2.next()) {
```

```
                        establishment.estname.setValue(rs2.getString(1),
rs2.getMetaData().getColumnType(2)); // fill establishments name

                        TransactionManager.setSuccess();

                    } else {

                        TransactionManager.getInstance().handleNotFound();

                    }

                    st2.close();

                // print employee and establishment

                LOGGER.info("Employee " + employee.ename + " works in establishment "
+ establishment.estname);


            }

            st.close();

        } catch(SQLException se) {

            TransactionManager.getInstance().handleError(se);

        }

    }
```

To get more info how to work with DB please read WORK WITH DB