1010data Java SDK User's Guide

(212) 405.1010 | info@1010data.com | Follow: @1010data | www.1010data.com

Contents

Introduction 3

Getting started 4

Basic usage 5
- Setting up an application 5
- Establishing a session 5
  - Login Types 6
  - Connecting via proxy 6
- Running a query 7
- Accessing query results 7
- Cleaning up a session 8

Exception handling 10

SAM pools and multi-threading 12
- Warm a SAM pool 14

Best practices 16

Updating existing applications 17

Reference 20

Troubleshooting and support 21
- Support requests 21

Introduction

The 1010data Java SDK enables developers to create applications that connect their applications to the 1010data Insights Platform for processing and retrieval of big data.

This guide contains the information you need to start developing a new application if this is your first time working with the SDK. The Java SDK enables the development of both single-threaded and multithreaded applications, working with either single-user Insights Platform sessions or SAM pools.

The following sections may be particularly helpful:

- *Getting started* on page 4 contains the information you need to start using the Java SDK. It also includes instructions on how to try out the examples provided with the library files.
- *Basic usage* on page 5 walks you through a basic workflow to create a single-threaded, single-user application.
- *SAM pools and multi-threading* on page 12 explains SAM pool logins and multithreaded applications.

For developers who are more familiar with our SDKs and developing on the Insights Platform, there is also reference material and lots of examples to answer your questions.

If you need information about a specific class or function, refer to the *1010data Java SDK Reference*.

Getting started

Before you start writing your first application that connects to the 1010data Insights Platform, you need to download the Java SDK files.

Requirements

The 1010data Java SDK is available for Windows, MacOS, and Linux. You must have Java 1.6 or higher installed on your system.

Installation

You can download the zip file for the Java SDK from *https://www2.1010data.com/downloads/tools/java2/JavaSDK1010v2.zip*. It contains three folders, `bin`, `doc`, and `examples`. The `bin` folder contains the SDK library files in the `com` folder.

Unzip the `JavaSDKv2` zip file and add the `com` directory into a folder on your Java `CLASSPATH` or into your working directory.

Example projects

The `examples` folder contains the following examples:

AllTypes

Casts Insights Platform data types (text, integer, decimal, bigint, date, time, and date+time) into their appropriate Java types.

ColumnWriter

Prints the columns of a table one at a time.

RowWriter

Prints the rows of a table one at a time.

SimpleThread

Uses two threads to create two session objects, retrieves information from two tables, and downloads the data into two separate files.

TTPager

Displays a complex web server example using multiple threads.

Basic usage

Creating a new application with the 1010data Java SDK includes setting up your application framework, establishing a session, submitting a query, receiving results, and releasing the session.

Setting up an application

Your application setup includes importing any required libraries, including the 1010data Java SDK and creating a class that holds the functions that interact with the Insights Platform and the `Main` method.

In the example, the `DocumentationExample` class contains the `Main` method and `printRows` function. The `printRows` function retrieves each row and prints it.

```
import com.tentendata.javasdk1010v2.*;

public class DocumentationExample {
    public static void printRows(Row r) {
        System.out.println(r);
    }

    public static void Main() {
    }
}
```

Establishing a session

A 1010data Insights Platform session is an instance of a particular user ID logged into the platform within a certain environment. A session is comprised of a dedicated set of resources that can serve a single request at a time.

A single platform session may have multiple `Session` objects attached to it. When a user logs into the platform, if no session is active, a new one is created. If a session is currently active for that user, the login type determines if that session is possessed, ended and a new one is started, or fails to connect. Currently, each user ID can only have one platform session associated with it. For more information about login type, see *Login Types* on page 6.

In the example, the `Session` object is created in the `Main` method. The new `Session` object `testSession` takes four parameters:

- a URI (`gateway`)
- a valid Insights Platform user name (`user`)
- that user's password (`pwd`)
- the login type, which is `LoginType.POSSESS` in this example

If you are connecting via a proxy server, your gateway string contains your proxy server address and any credentials. For more information, see *Connecting via proxy* on page 6.

```
import com.tentendata.javasdk1010v2.*;

public class DocumentationExample {
    public static void printRows(Row r) {
        System.out.println(r);
    }

    public static void main(String[] args) {
        String gateway = "https://www2.1010data.com/cgi-bin/gw.k";
        String user = "[USER_NAME]";
        String pwd = "[USER_PASSWORD]";
```

```
        Session testSession = new Session(gateway, user, pwd,
                                          LoginType.POSSESS);
    }
}
```

Note: *[USER_NAME]* and *[USER_PASSWORD]* are placeholders for valid Insights Platform user name and password.

Login Types

The login type determines what happens when the application establishes a connection to the Insights Platform and another session is currently active.

There are three valid login types: `LoginType.POSSESS`, `LoginType.KILL`, and `LoginType.NOKILL`.

- `LoginType.POSSESS` attaches the new connection to an existing session if one is active when the connection is established.
- `LoginType.KILL` ends an existing session and starts a new one when the connection is established.
- `LoginType.NOKILL` fails to connect if an existing session is active when the connection is established.

If no session is active, a new session is started.

Connecting via proxy

If your application is connecting to the Insights Platform via a proxy server, your gateway string needs to be modified to include the proxy server's address (*MYCORPORATEPROXYADDRESS*)

Depending on your proxy server's configuration, you may also need to include:

- user name (*PROXY_USERNAME*)
- password (*PROXY_PASSWORD*)
- port (*PORT*)

The proxy server's information is added to the beginning of the gateway string between square brackets (`[]`).

```
[http://PROXY_USERNAME:PROXY_PASSWORD@
MYCORPORATEPROXYADDRESS:PORT]https://www2.1010data.com/cgi-bin/gw
```

If you're unsure of whether or not you need to specify a proxy server, or you need your proxy server credentials, contact your corporate technology support team or IT department.

Note: This example does not use square brackets (`[]`) to denote variables.

Examples

Some proxy servers will require a full set of credentials, for example:

```
[http://aHamilton:g3n3ricPWD@my.corporate.proxy:8080]https://
www2.1010data.com/cgi-bin/gw
```

A proxy server may only require a user name, for example:

```
[http://hGranger@my.corporate.proxy:8080]https://www2.1010data.com/cgi-bin/gw
```

A proxy server may not require any credentials or a port number, for example:

```
[http://my.corporate.proxy]https://www2.1010data.com/cgi-bin/gw
```

Running a query

Your application should act as an intermediary between your users and the platform to pass in a query and receive the queried data.

Gathering specific sets of data or scoping your data should occur within queries. Typically, the analytical work is done by the Macro Language query specified by the `Query` object. For more information on writing queries using the 1010data Insights Platform Macro Language, see the *1010data Reference Manual*.

Use the `Query` method `run` to run queries.

In the example, the `Query` object is created in the `Main` function. The new `Query` object `exampleQuery` takes the path to the table that the query is being run against and the string containing the Macro Language query and the path to the table to which the query is applied. This Macro Language query selects the first 10 rows of the table. As a best practice, longer queries should be saved in a separate file and read.

```
import com.tentendata.javasdk1010v2.*;

public class DocumentationExample {

    public static void printRows(Row r) {
        System.out.println(r);
    }

    public static void main(String[] args) {
        String gateway = "https://www2.1010data.com/cgi-bin/gw.k";
        String user = "[USER_NAME]";
        String pwd = "[USER_PASSWORD]";
        String ops = "<sel value=\"(between(i_;1;10))\"/>";
        Session testSession = new Session(gateway, user, pwd,
                                          LoginType.POSSESS);
        Query exampleQuery = new Query(testSession, path, ops);
        ResultSet results = exampleQuery.run();
    }
}
```

Note: *[USER_NAME]* and *[USER_PASSWORD]* are placeholders for valid Insights Platform user name and password.

Accessing query results

After querying the 1010data Insights Platform, your application receives a set of results, which are accessible through the `ResultSet` object.

In this example, the `ResultSet` object is set to the result of `exampleQuery.run()`. This application prints the rows in the `ResultSet`.

```
import com.tentendata.javasdk1010v2.*;

public class DocumentationExample {

    public static void printRows(Row r) {
        System.out.println(r);
    }

    public static void main(String[] args) {
        String gateway = "https://www2.1010data.com/cgi-bin/gw.k";
        String user = "[USER_NAME]";
        String pwd = "[USER_PASSWORD]";
        String ops = "<sel value=\"(between(i_;1;10))\"/>";
```

```
        Session testSession = new Session(gateway, user, pwd,
                                          LoginType.POSSESS);
        Query exampleQuery = new Query(testSession, path, ops);
        ResultSet results = exampleQuery.run();
        for (long i = 0; i < results.numRows(); i++) {
            printRows(results.row(i));
        }
    }
}
```

Note: *[USER_NAME]* and *[USER_PASSWORD]* are placeholders for valid Insights Platform user name and password.

Cleaning up a session

Applications that use the 1010data Java SDK must manually clean up their 1010data Insights Platform sessions after use. You can do this using the `logout` method.

The `Session` class contains a `logout` method, which releases the session when it is invoked. Depending on the login type that was used to create the session, there are a few different ways the Insights Platform session might be cleaned up. If you are a single user logged in with `LoginType.POSSESS`, the platform session continues to run. If you're a single user logged in with `LoginType.KILL` or `LoginType.NOKILL`, the session is terminated. A SAM pool user's platform session is released back into the pool.

Cleaning up a platform session when it is no longer needed is particularly important if you have multiple users sharing a pool of IDs (SAM pool). The Java runtime does not guarantee that destructors are called when the program is terminated. Additionally, waiting for the garbage collector to clean up `Session` objects may result in a pool that appears fully utilized, but is waiting for `Session` objects to be destroyed. For more information about SAM pools, see *SAM pools and multi-threading* on page 12.

In this example, the `logout` method is used to clean up the session. To ensure that the session is cleaned up afterward, the `try`/`finally` statement is implemented with `testSession.logout();`. The `try`/`finally` idiom guarantees that when you're done with a session it gets properly deallocated.

```
import com.tentendata.javasdk1010v2.*;

public class DocumentationExample {

    public static void printRows(Row r) {
        System.out.println(r);
    }

    public static void main(String[] args) {
        String gateway = "https://www2.1010data.com/cgi-bin/gw.k";
        String user = "[USER_NAME]";
        String pwd = "[USER_PASSWORD]";
        String ops = "<sel value=\"(between(i_;1;10))\"/>";
        try {
            Session testSession = new Session(gateway, user, pwd,
                                              LoginType.POSSESS);
            Query exampleQuery = new Query(testSession, path, ops);
            ResultSet results = exampleQuery.run();
            for (long i = 0; i < results.numRows(); i++) {
                printRows(results.row(i));
            }
        }
        finally {
            testSession.logout();
        }
    }
}
```

Note: *[USER_NAME]* and *[USER_PASSWORD]* are placeholders for valid Insights Platform user name and password.

Exception handling

The Java SDK contains the `TenTenException` class.

The `TenTenException` class can be caught with `try`/`catch`. The `TenTenException` class is the parent class of all of the 1010data Java SDK exceptions. The following example uses `try`/`catch` for exception handling:

```
import java.io.IOException;
import java.io.InputStream;
import java.io.Writer;
import java.io.FileWriter;
import com.tentendata.javasdk1010v2.*;

public class RowWriter {
    public static void writerows(ResultSet results, Writer out)
    throws IOException {
        for (long i = 0; i < results.numRows(); i++) {
            Row row = results.row(i);
            Datum[] data = row.asArray();
            for (int j = 0; j < data.length; j++) {
                if (j > 0) {
                    out.write('\t');
                }
                out.write(data[j].toString());
            }
            out.write('\n');
        }
    }

    public static void main(String[] args) throws IOException {
        String url = "https://www2.1010data.com/cgi-bin/gw";
        String tablename = "pub.demo.weather.stations";
        String outfilename = "RowOutput.txt";
        String user;
        String pwd;
        String group;
        Session session = null;
        Query query;

        if (args.length < 2) {
            System.err.println("usage: RowWriter USERNAME PASSWORD");
            System.exit(1);
        }
        user = args[0];
        pwd = args[1];
        System.out.println("Single-user login for user " + user);
        try {
            session = new Session(url, user, pwd, LoginType.POSSESS);

            System.out.println("Session established.");
            query = new Query(session, tablename,
                              "<sel value=\"(elev>2)\"/>");
            ResultSet results = query.run();
            System.out.println("Query run.");
            FileWriter output = new FileWriter(outfilename);
            try {
                writerows(results, output);
            }
            finally {
                output.close();
```

```
            }
        }
        catch (TenTenException e) {
            System.err.println("1010 Error: " + e);
        }
        finally {
            if (session!=null) {
                session.close();
            }
        }
        System.out.println("Done.");
    }
}
```

SAM pools and multi-threading

A SAM pool enables a single set of credentials to be shared between client side threads to leverage multiple threads of parallelism on the 1010data Insights Platform.

Creating a new `Session` object for a SAM pool user is similar to creating a single-user `Session` object.

```
Session s = new Session(gateway, owner, pw, group, LoginType.POSSESS);
```

The SAM pool user session takes two different parameters. Instead of the individual user name `user`, this `Session` object takes the `owner` parameter. The `owner` parameter is the group owner's user name. The `group` parameter is the group ID. The SAM pool `Session` constructor takes the following parameter variables:

- `gateway` - a 1010data Insights Platform URI
- `group` - a group ID
- `owner` - the group owner ID
- `password` - the group owner password

If your users are using a SAM pool to log in, they will not have their own platform IDs. The users share a single set of credentials to log into the platform. A SAM pool has a number of IDs to use, but it is not limitless. You may have to account for cases where peak utilization creates a situation where more threads are trying to access the platform than your SAM pool is configured to handle.

The Java SDK is designed to queue during `Session` object construction when a SAM pool is fully utilized. During this queuing time the thread trying to construct a new `Session` object retries the request every ten seconds.

To ensure that your SAM pool is fully utilized, release sessions as soon as your thread is done prevents a full pool of IDs where all of the threads of database concurrency are in use at the same time. For more information, see *SAM pools and multi-threading* on page 12.

In this example, two threads construct two separate `Session` objects and log in with a SAM pool. The application retrieves data from two different tables and downloads them into two separate files.

```
import java.io.IOException;
import java.io.Writer;
import java.io.FileWriter;
import java.lang.Thread;
import com.tentendata.javasdk1010v2.*;

public class SimpleThread extends Thread {
    public String table = null;
    public String ops = null;
    private String URL, owner, password, group;
    private Writer output;

    public SimpleThread(String URL, String owner, String password,
                        String group, String table, String ops,
                        Writer output) {
        this.URL = URL;
        this.owner = owner;
        this.group = group;
        this.password = password;
        this.table = table;
        this.ops = ops;
        this.output = output;
    }

    public void run() {
        ResultSet results;
        Query query;
```

```
        Session session = null;
        try {
            session = new Session(URL, owner, password, group);
            System.out.println("Session started, username: " +
                               session.getUsername());
            query = new Query(session, table, ops);
            results = query.run();
            System.out.println("Ran query on " + table + ", returned " +
                               results.numRows() + " rows.");
            for (long i = 0; i < results.numRows(); i++) {
                Row row = results.row(i);
                Datum[] data = row.asArray();
                for (int j = 0; j < data.length; j++) {
                    if (j > 0) {
                        output.write('\t');
                    }
                    output.write(data[j].toString());
                }
                output.write('\n');
            }
            output.close();
        }
        catch (IOException e) {
            System.err.println("Error writing: " + e);
        }
        finally {
            if (session != null) {
                session.close();
            }
        }
    }

    public static void main(String[] args) throws IOException {
        String URL = "https://www2.1010data.com/cgi-bin/gw";
        String path1 = "pub.public_data.census.acs.all_acs_processed_tables";
        String path2 = "pub.public_data.census.acs.reference.column_names";
        String ops = "<sel value=\"(i_<50000)\"/>";

        if (args.length < 3) {
            System.out.println("Usage: SimpleThread OWNER PASSWORD GROUP");
            System.exit(1);
        }
        String owner = args[0];
        String password = args[1];
        String group = args[2];

        Thread thread1 = new SimpleThread(URL, owner, password, group,
                                          path1, ops,
                                          new FileWriter("Output1.txt"));
        Thread thread2 = new SimpleThread(URL, owner, password, group,
                                          path2, ops,
                                          new FileWriter("Output2.txt"));
        thread1.start();
        thread2.start();
        thread1.join();
        thread2.join();
    }
}
```

Warm a SAM pool

One way to reduce the amount of time it takes to retrieve results from the 1010data Insights Platform is to "warm" the SAM pool by logging in IDs ahead of time.

The `warmPool` function logs in available SAM pool user IDs. additionally, when warming the pool you can provide queries that are run on the IDs as they are logged in. This can speed the retrieval of results for users.

The `warmPool` function takes the following parameters:

- URI (`gateway`)
- Group owner name (`owner`)
- Group owner password (`password`)
- Group ID (`group`)
- List of `BaseQuery` objects (`queries`) - `BaseQuery` objects take a table name and 1010data Macro Language query, but are different from `Query` objects because they are not associated with a `Session` object. The default value is `None`.
- Path to a log file (`logfile`) - The default value is `None`.

Example without queries or a log file path and with them:

```
warmPool(gateway, owner, password, group)
```

```
warmPool(gateway, owner, password, group, queries, logfile)
```

You can run `warmPool` without logging in or from a session that is logged in, in which case, you will not have to re-input your information (`gateway`, `owner`, `group`, and `password`) again.

The used IDs should be released at the end of their activity with the platform. When they are released, they are not warmed automatically.

The example below uses `warmPool` to warm the SAM pool and `BaseQuery` to provide a query:

```
import com.tentendata.javasdk1010v2.*;

public class Warming {

    public static void main(String[] args) {
        String gateway;
        String queryXML;
        String tablePath;
        String username;
        String password;
        String group;
        Session session = null;
        Query query;

        BaseQuery[] queries;

        if (args.length < 3) {
            System.err.println("usage: PrintTable USERNAME PASSWORD GROUP");
            System.exit(1);
        }
        username = args[0];
        password = args[1];
        group = args[2];
        System.err.println("Group login to " +
                           group + ", owner " + username);

        gateway = "https://www2.1010data.com/cgi-bin/gw.k";
        queryXML = "<sel value=\"(us_postal_state_code='WA')\"/>";
        tablePath = "pub.public_data.fhfa.fhfa";
```

```
        queries = new BaseQuery[1];
        queries[0] = new BaseQuery(tablePath, queryXML);

        int w=Session.warmPool(gateway, username, password, group, queries,
                               "WarmLog.log");
        System.out.println("Warmed " + w + " uids");
        System.exit(8);

        try {
            session = new Session(gateway, username, password,
                                  LoginType.POSSESS);

            query = new Query(session, tablePath,
                              queryXML);
            query.setWindowSize(1000);
            ResultSet results = query.run();
            // Limit rows to 5000 for testing purposes.
            for (long i = 0; i < Math.min(5000,results.numRows()); i++) {
                Row row = results.row(i);
                Datum[] data = row.asArray();
                for (int j = 0; j < data.length; j++) {
                    if (j > 0) {
                        System.out.print('\t');
                    }
                    System.out.print(data[j].toString());
                }
                System.out.println();
            }
        }
        catch (TentenException exc) {
            System.err.println("1010 Exception occurred: " + exc.toString());
        }
        finally {
            if (session!=null) {
                session.close();
            }
        }
    }
}
```

Note: *USERNAME*, *GROUP*, and *PASSWORD* are placeholders for valid Insights Platform group owner name. group name, and password.

Best practices

There are several recommended best practices when using the 1010data Java SDK.

Clean up your sessions upon query completion

Use the `logout` method to clean up sessions. For more information, see *Cleaning up a session* on page 8.

Scroll through data sequentially

When accessing data, scrolling sequentially produces better performance results. You may want to filter or sort the data before scrolling through it to ensure sequential access.

Use XML queries for random access or set window size to 1

As a best practice, users should access random data via queries. The example query below requests a random 10% sample:

```
Query q = new Query(SESSION_NAME, PATH, "<sel value=\"draw_(33;10)\"/>");
```

If your users are going to ask for a random set of data intersections that do not progress incrementally without using a query, the best practice is to set your window size to 1. The window is the number of rows of data that buffered when the Java SDK requests results from the 1010data Insights Platform. By setting the window to 1, you retrieve each row individually. This prevents the system from retrieving larger sets of unneeded data at random points in the data set and prevents unnecessary transfer of data. In general, you won't have to worry about the window. It is worth noting that even with a window size of 1, this kind of access will most likely perform poorly.

If you do want to change the window size, you can use the `setWindowSize` method.

Have each thread create a `Session` object

In a multithreaded application, each thread should create its own `Session` object. If multiple threads that share a single `Session` object interact with the object in parallel, all but one of the threads will be queued. Users will experience much better performance if each thread has its own `Session` object. These objects can all be connected to the same Insights Platform session via `LoginType.POSSESS`.

Updating existing applications

The latest 1010data Java SDK is designed for developing applications in a style that is more closely compatible with Java development, specifically object-oriented programming.

Applications developed with the original 1010data Java SDK must be updated to use the object-oriented 1010data Java SDK. When you're updating your existing application, keep that applications developed for the original 1010data Java SDK:

- Required you to instantiate exactly one instance of the `JavaSDK1010` class. The object-oriented 1010data Java SDK does not require you to instantiate any instance of the `JavaSDK1010` class.
- Required you to synchronize multiple threads yourself. The object-oriented 1010data Java SDK, handles thread synchronization. As a best practice, each thread should create its own `Session` object. For more information, see *Best practices* on page 16 or *Establishing a session* on page 5.
- Required you to check for error codes after each method invocation. The object-oriented Java SDK uses an exception-based model. For more information, see *Exception handling* on page 10.
- Required applications using a SAM pool to manually acquire a UID via calls to `GetUID1010`. In the object-oriented 1010data Java SDK, this is not required. For more information, see *SAM pools and multi-threading* on page 12.

This example contains code written for the original 1010data Java SDK:

```
import com.tentendata.javasdk1010.*;

public class PrintTable {

    public static void main(String[] args) {
        JavaSDK1010 link;
        int login;
        String gateway;
        String queryXML;
        String tablePath;
        String username;
        String password;
        int queryID, block, windowSize, finRows, rowsPerBlock;
        long numRows;
        Object[][] colInfo, table;

        if (args.length < 2) {
            System.err.println("usage: PrintTable USERNAME PASSWORD");
            System.exit(1);
        }
        username = args[0];
        password = args[1];
        System.err.println("Single-user login for user " + username);

        gateway = "https://www2.1010data.com/cgi-bin/gw";
        queryXML = "<sel value=\"(year&gt;1900)\"/>";
        tablePath = "pub.demo.baseball.batting";
        windowSize = 10000;
        block = 40000;
        finRows = 0;

        // Create exactly one JavaSDK1010 object.
        link = new JavaSDK1010();
        login = link.login1010(gateway, username, password, 1);
        // We need to check the return code after every action to
        // make sure that it succeeded.
        if (login < 0) {
            System.err.println("Failed " + login + ": " +
```

```
                                link.msg1010(login));
            System.exit(1);
        }
        try {
            queryID = link.newquery1010();
            if(queryID < 0) {
                System.err.println("Fail to create query: (" +
                                   link.rc1010(login) + ") " +
                                   link.msg1010(login));
                System.exit(1);
            }
            // -2 is COMPRESSED_BINARY
            link.querymode1010(queryID, windowSize, -2);
            link.prepquery1010(queryID, tablePath, queryXML);

            // Run the query.
            if(link.runquery1010(login, queryID) < 0) {
                System.err.println("Failed to run query: (" +
                                   link.rc1010(login) + ") " +
                                   link.msg1010(login));
                System.exit(1);
            }
            colInfo = link.querycols1010(queryID);
            if(colInfo == null) {
                System.err.println("Failed to get column info: (" +
                                   link.rc1010(login) + ") " +
                                   link.msg1010(login));
                System.exit(1);
            }

            numRows = link.queryrows1010(queryID);

            // For testing, limit the number of rows to 5000.
            numRows = Math.min(5000, numRows);
            block = 1000;

            while (finRows < numRows) {
                // Pull down one block at a time.
                // Start at finRows and get all the columns.
                table = link.getdataRM1010(queryID, finRows +
                                           block, colInfo);
                if(table == null) {
                    System.err.println("Failure at getting data: (" +
                                   link.rc1010(login) + ") " +
                                   link.msg1010(login));
                    System.exit(1);
                }
                finRows += block;

                // For each row in this block...
                for (int i = 0; i < table.length; i++) {
                    // for each column in this row...
                    for (int j = 0; j < table[i].length; j++) {
                        System.out.print(table[i][j] + "\t");
                    }
                    System.out.println();
                }
                // It is important to set the table to null
                // when you are finished using it.
                table = null;
            }
        }
        finally {
```

```
            link.logout1010(login);
        }
    }
}
```

This example contains code written for the object-oriented 1010data Java SDK:

```
import com.tentendata.javasdk1010v2.*;

public class PrintTable {

    public static void main(String[] args) {
        String gateway;
        String queryXML;
        String tablePath;
        String username;
        String password;
        Session session = null;
        Query query;

        if (args.length < 2) {
            System.err.println("usage: PrintTable USERNAME PASSWORD");
            System.exit(1);
        }
        username = args[0];
        password = args[1];
        System.err.println("Single-user login for user " + username);

        gateway = "https://www2.1010data.com/cgi-bin/gw";
        queryXML = "<sel value=\"(year&gt;1900)\"/>";
        tablePath = "pub.demo.baseball.batting";

        try {
            session = new Session(gateway, username, password,
 LoginType.POSSESS);

            query = new Query(session, tablePath,
                              queryXML);
            query.setWindowSize(1000);
            ResultSet results = query.run();
            // Limit rows to 5000 for testing purposes.
            for (long i = 0; i < Math.min(5000,results.numRows()); i++) {
                Row row = results.row(i);
                Datum[] data = row.asArray();
                for (int j = 0; j < data.length; j++) {
                    if (j > 0) {
                        System.out.print('\t');
                    }
                    System.out.print(data[j].toString());
                }
                System.out.println();
            }
        }
        catch (TentenException exc) {
            System.err.println("1010 Exception occurred: " + exc.toString());
        }
        finally {
            if (session!=null) {
                session.close();
            }
        }
    }
}
```

Reference

The *1010data Java SDK Reference* contains the details about its classes and functions.

For more information, refer to the *1010data Java SDK Reference*.

Troubleshooting and support

You can try to troubleshoot issues with the Java SDK. If you cannot resolve the issue, file a support request.

Enter a proxy URL if your organization requires one

If you receive the error message `Transport error 7: Couldn't connect to server`, it means that your web browser is configured to use your corporate proxy, but your operating system is not. Contact your company's IT staff to obtain your proxy's URL or IP address, what port it listens on, and if it requires a username and password. Then follow the instructions in the *Connecting via proxy* on page 6 section of this guide.

If you still cannot resolve your issue, you can submit a support request. For more information, see *Support requests* on page 21.

Support requests

If you are using the 1010data SDKs, 1010data offers full support.

Including information such as the log file generated by the Java SDK or the source code containing the problems enables 1010data Support to respond more effectively to your request.

Include a log file

You must submit a log file with your support request. This SDK includes a feature to turn on logging. Use the `Session` constructor that includes the `logfile` parameter. Specify the `File` object and reproduce the issue with logs. Include these logs in your support request.

```
Session s = new Session(gateway, user, pwd, LoginType.POSSESS, logfile);
```

The `logfile` specifies the path of where to write the log file.

Include additional information

To expedite the support process, include any:

- Runtime error messages
- Stack traces
- Core dumps
- Exceptions

Include source code

Creating or isolating standalone source code is not always feasible, however sending in source code greatly expedites the support process. Wherever possible, send reproducible source code that can be run and tested.