# **Connecting to an Ethereum client with Java, Eclipse and Web3j**

**Other articles in this series:** - [Manage an Ethereum account with Java and Web3j](https://kauri.io/#article/925d923e12c543da9a0a3e617be963b4) - [Generate a Java Wrapper from your Smart Contract](https://kauri.io/#article/84475132317d4d6a84a2c42eb9348e4b) - [Interacting with an Ethereum Smart Contract in Java](https://kauri.io/#article/14dc434d11ef4ee18bf7d57f079e246e) - [Listening for Ethereum Smart Contract Events in Java](https://kauri.io/#article/760f495423db42f988d17b8c145b0874) - [Using Pantheon, the Java Ethereum Client with Linux](https://kauri.io/#article/276dd27f1458443295eea58403fd6965)

https://kauri.io/#article/b9eb647c47a546bc95693acc0be72546

[**Ethereum**](https://www.ethereum.org/) is a Blockchain, which means it operates on a [peer-to-peer network](https://en.wikipedia.org/wiki/Peer-to-peer) composed of thousand of nodes where each node agrees on the next state.

In order to interact with the Ethereum global state (distributed database), a program needs to connect to a node that exposes the standard [JSON-RPC API](https://github.com/ethereum/wiki/wiki/JSON-RPC#json-rpc-api) which can be used to execute operations on the Ethereum blockchain.

In this article, we will learn how to start an Ethereum Java project and connect to a node using the Java library [**Web3j**](https://web3j.io/), a lightweight and modular library implementing all the functionallities required to work with Ethereum (JSON-RPC API client, wallet account management, Java Smart Contract wrapper, support for ENS, ERC20 and ERC721 and much more).

### **Prerequisite**

To run this tutorial, we must have the following installed:

* [Java programming language](https://java.com/en/download/) (> 8)

$ java -**version**

java **version** "1.8.0\_201"

* A package and dependancy manager, for example [Maven](https://maven.apache.org/) or [Gradle](https://gradle.org/install/)
* An IDE (Integrated development environment), for this tutorial, we use [Eclipse](https://www.eclipse.org/downloads/)

### **Start a new project**

First create a new Maven project called java\_ethereum in Eclipse.

#### 1. Create a new Maven project

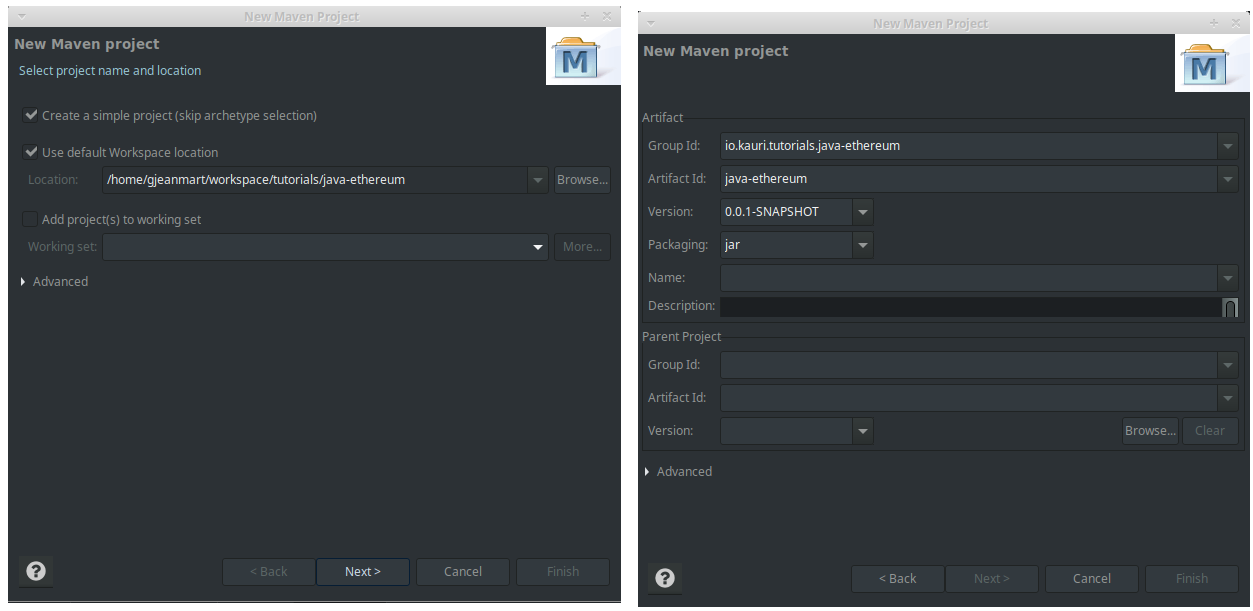
Once Eclipse is launched, we need to create a new Maven project. Go to File > New > Project > Maven > Maven Project

Check the box Create a simple project (skip archetype selection) and click on Next >.

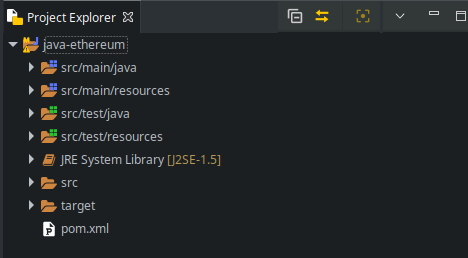
Next screen, enter the Group ID and Artifact ID of our project then click Finish.

Group Id: io.kauri.tutorials.java-ethereum

Artifact Id: java-ethereum



It should result of a new project in the Project Explorer



#### 2. Configure our project to use Java 8

Finally, we need to tell Eclipse and Maven to use Java version 8.

Edit the file pom.xml and add the following lines before </project>

<properties>

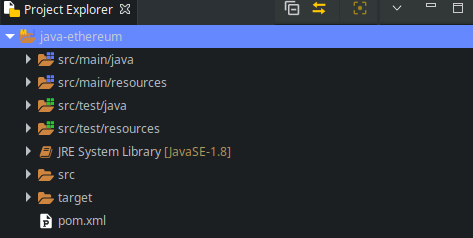
<maven.compiler.target>1.8</maven.compiler.target>

<maven.compiler.source>1.8</maven.compiler.source>

</properties>

Now, right click on the project name in the Project Explorer and click on Maven > Update Project. Click OK in the dialog box that pops up.

In the Project Explorer, You should see the JRE System library changing from **JavaSE-1.5** to **JavaSE-1.8**.



### **Add Web3j library to our project**

In this step, we import the latest version of Web3j to our project via maven.

In Eclipse, edit the file pom.xml and add the following lines before </project>:

<dependencies>

<dependency>

<groupId>org.web3j</groupId>

<artifactId>core</artifactId>

<version>4.3.0</version>

</dependency>

</dependencies>

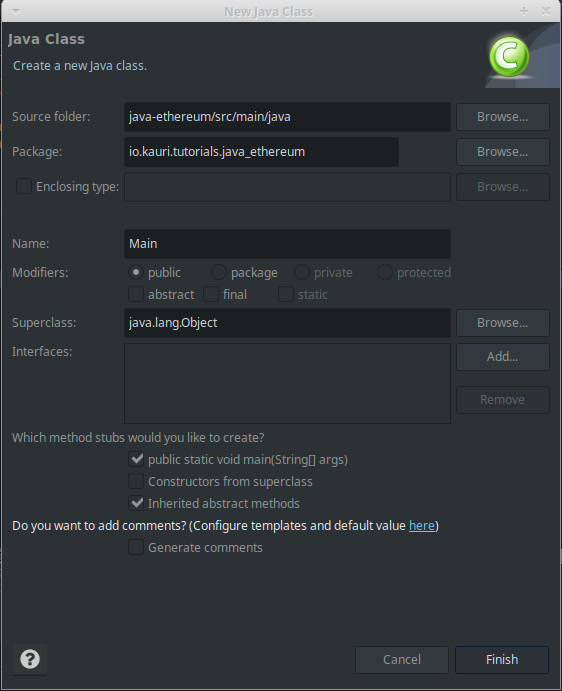
Full pom.xml file available [*here*](https://github.com/gjeanmart/kauri-content/blob/master/java-ethereum/pom.xml)

Save file and dependencies will import. In your package explorer you will see a Maven dependencies folder with all the JAR (Java ARchive) packages for web3j and its dependencies.

### **Create a Main class**

Now, we have all the required dependencies to use Web3j, we can start coding our Ethereum Java program.

Create a Java class Main.java in your project by right-clicking on the project and selecting New > Class. Enter the package name io.kauri.tutorials.java\_ethereum, the class name Main and check public static void main(String[] args).



Click on Finish to generate the skeleton file.

*//Main.java*

**package** io.kauri.tutorials.java\_ethereum;

**public** **class** **Main** {

**public** **static** **void** **main**(String[] args) {

*// TODO Auto-generated method stub*

}

}

### **Connect to an Ethereum node with Web3j.**

Now we have created our project, imported the Web3j library and prepared a program to run our code. We can now connect to an Ethereum node and start executing operations over the JSON-RPC API abstracted by Web3j.

#### 1. Add imports

First import the packages needed for our code, or allow your IDE to automatically import them for you:

**import** java.io.IOException;

**import** org.web3j.protocol.Web3j;

**import** org.web3j.protocol.http.HttpService;

**import** org.web3j.protocol.core.methods.response.EthBlockNumber;

**import** org.web3j.protocol.core.methods.response.EthGasPrice;

**import** org.web3j.protocol.core.methods.response.Web3ClientVersion;

#### 2. Connect to the node

To connect to the node, Web3j requires the JSON-RPC API endpoint:

Web3j web3 = Web3j.build(**new** HttpService("<NODE ENDPOINT>"));

##### **Local Ethereum node or ganache-cli**

If you are running locally a [Geth](https://geth.ethereum.org/), [Parity](https://www.parity.io/), [Pantheon](https://github.com/PegaSysEng/pantheon) client or [ganache-cli](https://github.com/trufflesuite/ganache-cli). Your node JSON-RPC API endpoint is http://localhost:8545 by default

Web3j web3 = Web3j.build(**new** HttpService("http://localhost:8545"));

##### **Ganache application: Local development blockchain**

If you are running the [Ganache](https://www.trufflesuite.com/ganache) application on your machine. Your node JSON-RPC API endpoint is http://localhost:7545 by default. ganche-cli uses port 8545

Web3j web3 = Web3j.build(**new** HttpService("http://localhost:7545"));

Note: As a test network, Ganache doesn't support all the JSON-RPC API operations specified, for example *net\_peercount*.

##### **Infura: Hosted nodes for public mainet and testnets**

If you use [Infura](https://infura.io/). The node JSON-RPC API endpoint is https://<network>.infura.io/v3/<project key>.

Web3j web3 = Web3j.build(**new** HttpService("https://mainnet.infura.io/v3/<project key>"));

#### 3. Execute API operations

Web3j implements a JSON-RPC API client for Ethereum which can be used in the following way <response> = web3.<operation>.send(). For example:

**try** {

*// web3\_clientVersion returns the current client version.*

Web3ClientVersion clientVersion = web3.web3ClientVersion().send();

*//eth\_blockNumber returns the number of most recent block.*

EthBlockNumber blockNumber = web3.ethBlockNumber().send();

*//eth\_gasPrice, returns the current price per gas in wei.*

EthGasPrice gasPrice = web3.ethGasPrice().send();

} **catch**(IOException ex) {

**throw** **new** RuntimeException("Error whilst sending json-rpc requests", ex);

}

**Note:** Serilization of the JSON-RPC request can raise an IOException exception, so you need to handle it.

### **Result**

The following code shows the entire Java program which connects to an Ethereum node and runs some JSON-RPC calls.

*//Main.java*

**package** io.kauri.tutorials.java\_ethereum;

**import** java.io.IOException;

**import** org.web3j.protocol.Web3j;

**import** org.web3j.protocol.core.methods.response.EthBlockNumber;

**import** org.web3j.protocol.core.methods.response.EthGasPrice;

**import** org.web3j.protocol.core.methods.response.Web3ClientVersion;

**import** org.web3j.protocol.http.HttpService;

**public** **class** **Main** {

**public** **static** **void** **main**(String[] args) {

System.out.println("Connecting to Ethereum ...");

Web3j web3 = Web3j.build(**new** HttpService("http://localhost:8545"));

System.out.println("Successfuly connected to Ethereum");

**try** {

*// web3\_clientVersion returns the current client version.*

Web3ClientVersion clientVersion = web3.web3ClientVersion().send();

*// eth\_blockNumber returns the number of most recent block.*

EthBlockNumber blockNumber = web3.ethBlockNumber().send();

*// eth\_gasPrice, returns the current price per gas in wei.*

EthGasPrice gasPrice = web3.ethGasPrice().send();

*// Print result*

System.out.println("Client version: " + clientVersion.getWeb3ClientVersion());

System.out.println("Block number: " + blockNumber.getBlockNumber());

System.out.println("Gas price: " + gasPrice.getGasPrice());

} **catch** (IOException ex) {

**throw** **new** RuntimeException("Error whilst sending json-rpc requests", ex);

}

}

}

Full file available [*here*](https://github.com/gjeanmart/kauri-content/blob/master/java-ethereum/src/main/java/io/kauri/tutorials/java_ethereum/Main.java)

To run the program, right-click on the file Main.java and click on Run As > Java Application. You should see in the console the following result.

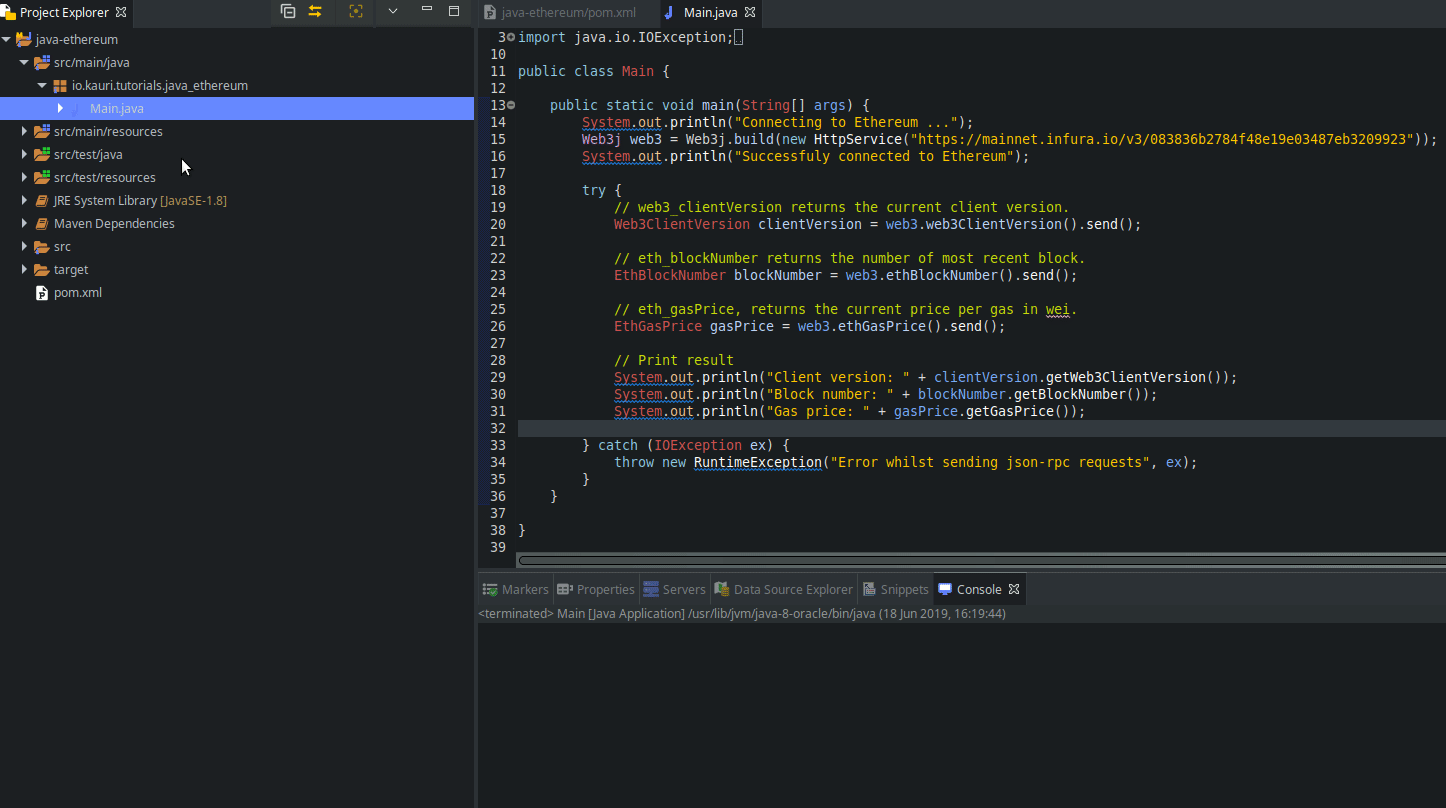
Connecting to Ethereum ...

Successfuly connected to Ethereum

Client version: Geth/v1.8.22-omnibus-260f7fbd/linux-amd64/go1.11.1

Block number: 7983049

Gas price: 3000000000



……………………………………………

**package** io.kauri.tutorials.java\_ethereum;

**import** java.io.IOException;

**import** org.web3j.protocol.Web3j;

**import** org.web3j.protocol.http.HttpService;

**import** org.web3j.protocol.core.methods.response.EthBlockNumber;

**import** org.web3j.protocol.core.methods.response.EthGasPrice;

**import** org.web3j.protocol.core.methods.response.Web3ClientVersion;

**public** **class** Main {

**public** **static** **void** main(String[] args) {

Web3j web3 = Web3j.*build*(**new** HttpService("https://mainnet.infura.io/v3/9e3750265c40488c8fd61752d883f7b2"));

**try** {

// web3\_clientVersion returns the current client version.

Web3ClientVersion clientVersion = web3.web3ClientVersion().send();

//eth\_blockNumber returns the number of most recent block.

EthBlockNumber blockNumber = web3.ethBlockNumber().send();

//eth\_gasPrice, returns the current price per gas in wei.

EthGasPrice gasPrice = web3.ethGasPrice().send();

System.***out***.println("Client version: " + clientVersion.getWeb3ClientVersion());

System.***out***.println("Block number: " + blockNumber.getBlockNumber());

System.***out***.println("Gas price: " + gasPrice.getGasPrice());

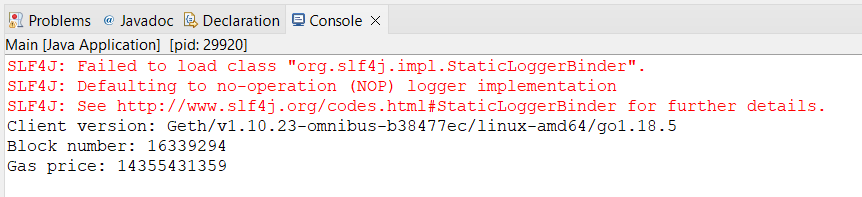
} **catch**(IOException ex) {

**throw** **new** RuntimeException("Error whilst sending json-rpc requests", ex);

}

}

}



…………………………………………………………..