Università di Pisa

AI-DII

Hands-on session 1: Apache Maven & Jinterface

Distributed Systems and Middleware Technologies

2023/2024

By José Luis Corcuera Bárcena

Agenda

1. What is Maven?
2. The POM
3. Maven advance topics
4. Enabling Java - Erlang communication - Jinterface
5. Jinterface: starting a node
6. Jinterface: sending messages
7. Jinterface: receiving messages
8. Getting ready our development environment
9. Exercises

Millions of lines of code

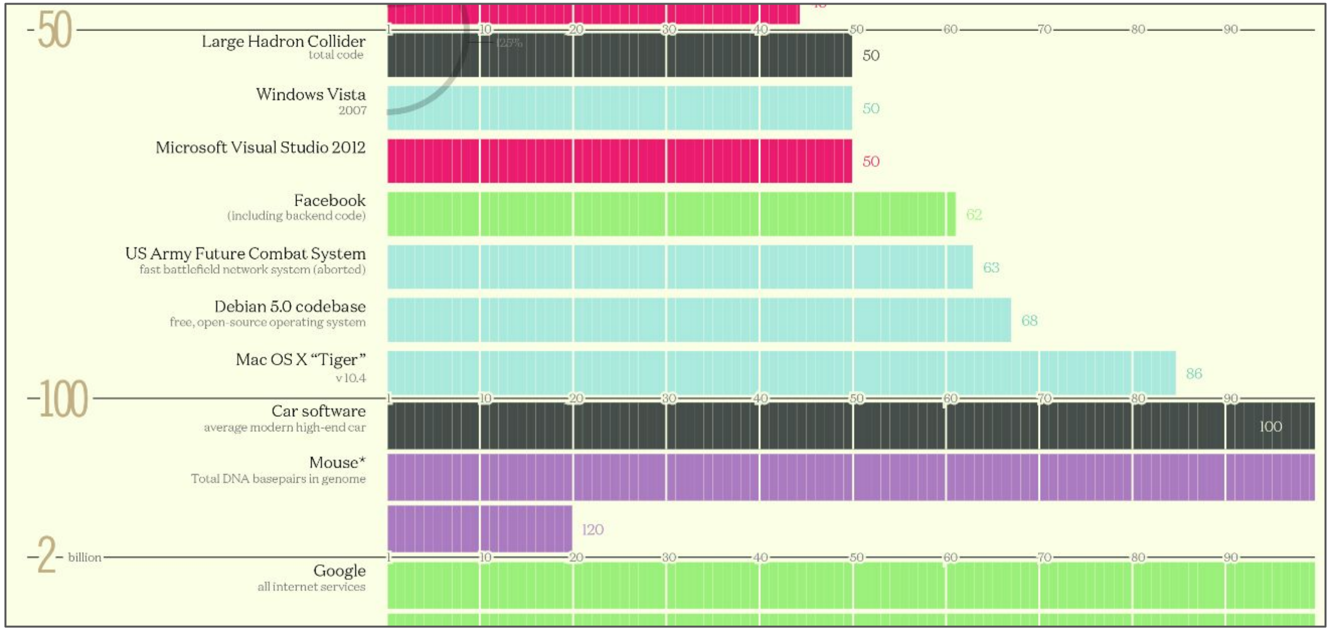


Image from: https://informationisbeautiful.net/visualizations/million-lines-of-code/

How do software development teams build the final executable?

What is Maven?

- Maven comes from a Yiddish word, *meaning accumulator of knowledge*.

- Maven is a

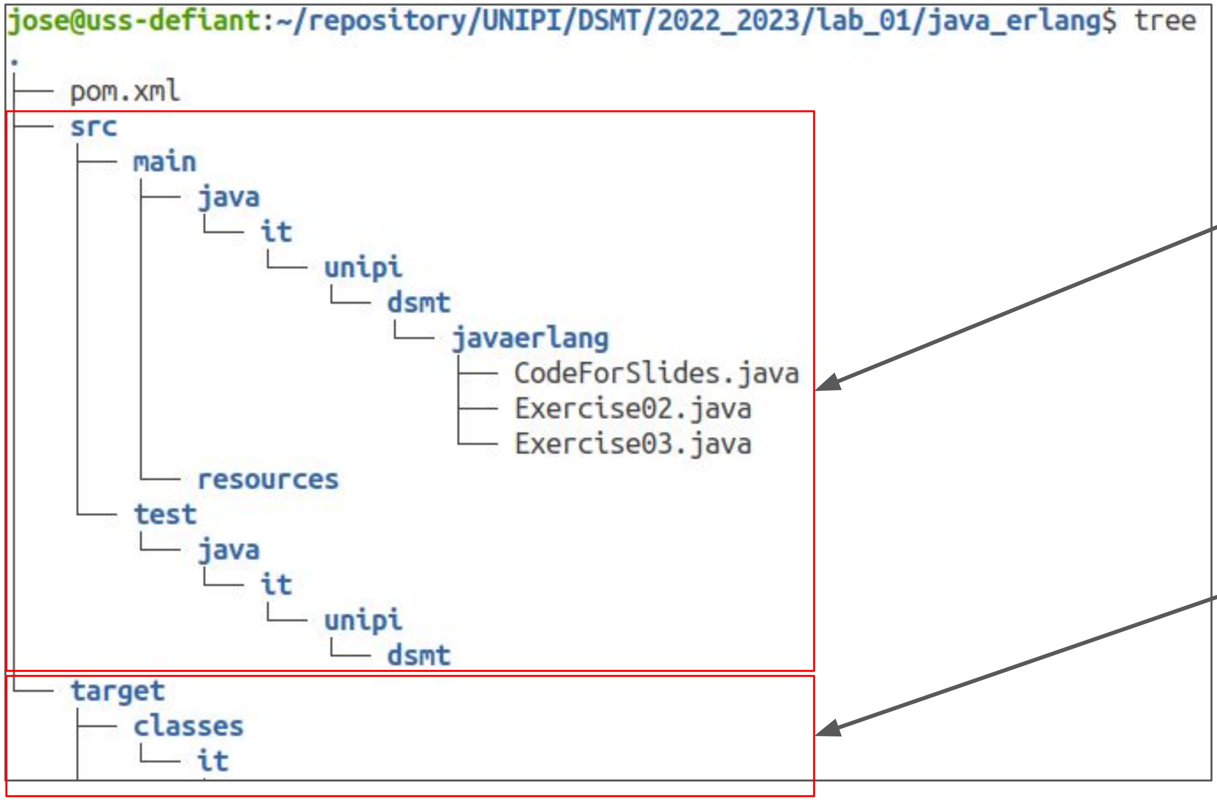
  BUILD TOOL

  DEPENDENCY MANAGEMENT TOOL

  DOCUMENTATION TOOL

- Formally is a **project management tool**, which encompasses a project object model, a set of standards, a project lifecycle, a dependency management system and logic for executing plugin goals at defined phases in a lifecycle.

Standardised directory layout



```
jose@uss-defiant:~/repository/UNIPI/DSMT/2022_2023/lab_01/java_erlang$ tree
.
├── pom.xml
├── src
│   ├── main
│   │   ├── java
│   │   │   └── it
│   │   │       └── unipi
│   │   │           └── dsmt
│   │   │               └── javaerlang
│   │   │                   ├── CodeForSlides.java
│   │   │                   ├── Exercise02.java
│   │   │                   └── Exercise03.java
│   │   └── resources
│   └── test
│       └── java
│           └── it
│               └── unipi
│                   └── dsmt
└── target
    ├── classes
    │   └── it
```

The **src** folder contains all the resources to be used during the building of your project.

This directory is used to generate intermediate files (i.e.: class files) and the result of the build process.

The POM

- Maven is based on the concept of project object model (POM).
- XML file, always residing in the base directory of the project as **pom.xml**. Users defined POMs extend the Super POM.
- The POM contains information about the project and various configuration detail used by Maven to build the project.
- The Maven POM is **declarative**, no procedural details are needed.

POM Structure

The POM contains four categories of description and configuration:

- General project information

  human readable information

- Build Settings

  add plugins, attach plugin goal to lifecycle

- Build Environment

  describe the “family” environment in which Maven lives

- POM Relationships

  coordinates, inheritance, aggregations, dependencies

An example of POM

```
<?xml version="1.0" encoding="UTF-8" ?>
<project xmlns="http://maven.apache.org/POM/4.0.0"
        xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
        xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
        http://maven.apache.org/xsd/maven-4.0.0.xsd" >

 <modelVersion>4.0.0</modelVersion>
 <groupId>it.unipi.dsmt.javaerlang</groupId>
 <artifactId>java erlang</artifactId>
 <version>1.0.0-SNAPSHOT</version>

 <name>${artifactId}-${version}</name>

 <dependencies>
   <dependency>
     <groupId>com.ericsson</groupId>
     <artifactId>erlang-jinterface</artifactId>
     <version>1.13.1</version>
   </dependency>
 </dependencies>

 <properties>
   <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>
   <maven.compiler.source>11</maven.compiler.source>
   <maven.compiler.target>11</maven.compiler.target>
 </properties>

 <build>
     <finalName>${artifactId}</finalName>
 </build>
</project>
```

General project information.

Project coordinate = groupId + artifactId + version

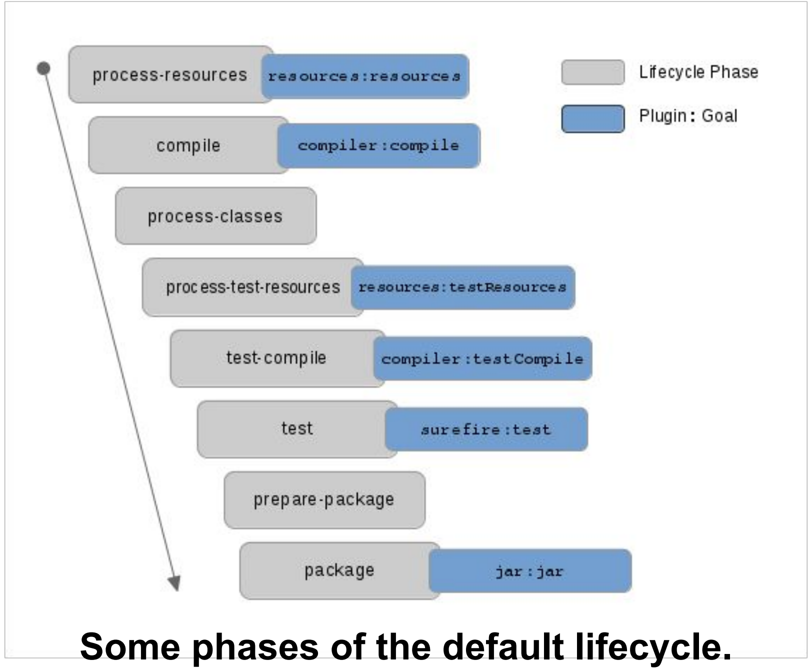
The Maven trinity!

POM Relationships.

Build settings and environment section here.

Maven lifecycle

- A lifecycle is a organised sequence of phases that give order to a sequence of goals.
- **Lifecycle phases don't come with any functionality; to carry out a task they rely on plugins.**
- Goals are packaged in plugins that are tied to phases.
- Three Standard Lifecycle:
  - Clean
  - default (or build)
  - Site



Some phases of the default lifecycle.

Image from: https://www.codetab.org/tutorial/apache-maven/maven-lifecycle-goals/

Maven repository

- Directory of packaged JAR files with pom.xml.
- Maven searches for dependencies in the repositories.

  Three types of maven repository exists:

  - Local Repository
  - Central Repository
  - Remote Repository

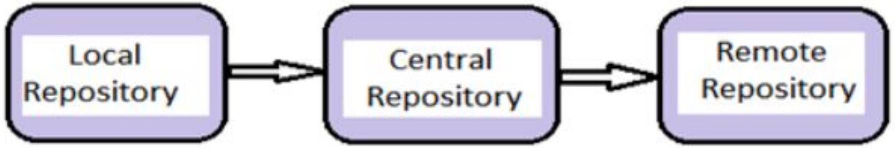


Image from: https://www.w3schools.blog/explain-maven-repository-search-order

- Downloaded dependencies are stored in the Local Repository.
- The local repository is located at $HOME/.m2 folder.

Enabling Java-Erlang communication - Jinterface (1)

- **Java library for communicating with Erlang processes.**

- **Two set of classes**:
  - those that provide the actual communication and
  - those that provide Java representation of Erlang data types.

- Erlang data types represented in Java extends the parent class **OtpErlangObject** and they are identified by the **OtpErlang prefix**.

Enabling Java-Erlang communication - Jinterface (2)

- **Maven repositories does not have latest version of Jinterface, however, Erlang distributions include this connector.**
- In order to install manually into your local Maven repository this dependency, please run the following command in a terminal:

```
mvn install:install-file \
  -Dfile=/usr/lib/erlang/lib/jinterface-1.13/priv/OtpErlang.jar \
  -DgroupId=com.ericsson \
  -DartifactId=erlang-jinterface \
  -Dversion=1.13.1 \
  -Dpackaging=jar \
  -DgeneratePom=true
```

DO NOT forget to update the path of this jar file and the version.

Jinterface: starting a node

```
import com.ericsson.otp.erlang.*;

…

String cookie = "abcde";

String nodeId = "client_node@127.0.0.1";

String mailBoxId = "MyMailBox";

OtpNode node = new OtpNode(nodeId, cookie);

OtpMbox mailBox = node.createMbox(mailBoxId);
```

Creation of an Erlang node and setting an alias to a process mailbox.

Jinterface: sending messages (1)

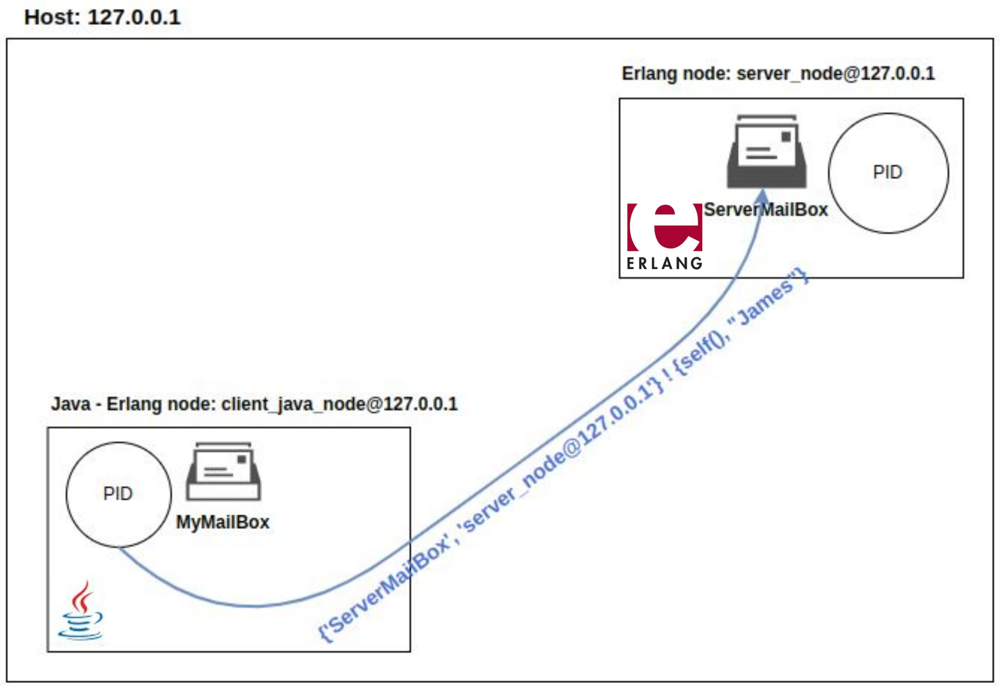
```
import com.ericsson.otp.erlang.*;

…

String serverMailBoxId = "ServerMailBox";

String serverNodeId = "server_node@127.0.0.1";


OtpErlangObject[] content = new OtpErlangObject[2];

content[0] = mailBox.self();

content[1] = new OtpErlangString("James");

OtpErlangTuple message = new OtpErlangTuple(content);


mailBox.send(serverMailBoxId, serverNodeId, message);
```

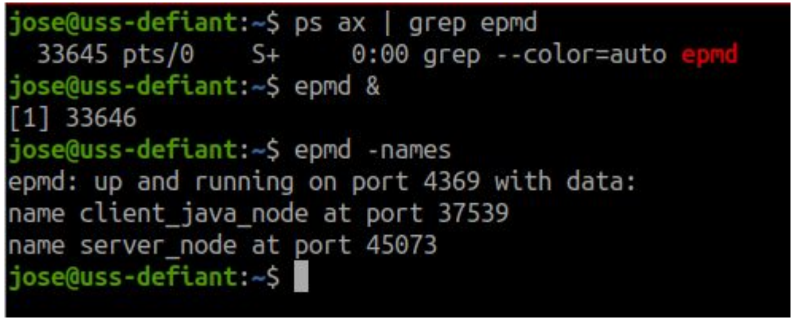
In this example we want to send a Tuple of two elements to an Erlang process.

Note that the PID value is sent.

Since we don't have the PID of the remote process, we have to make use of its MailBoxId and NodeId.

Jinterface: sending messages (2)


Host: 127.0.0.1
Erlang node: server_node@127.0.0.1
ServerMailBox
PID
ERLANG
Java - Erlang node: client_java_node@127.0.0.1
PID
MyMailBox
{'ServerMailBox', 'server_node@127.0.0.1'} ! {self(), "James"}



```
jose@uss-defiant:~$ ps ax | grep epmd
 33645 pts/0    S+     0:00 grep --color=auto epmd
jose@uss-defiant:~$ epmd &
[1] 33646
jose@uss-defiant:~$ epmd -names
epmd: up and running on port 4369 with data:
name client_java_node at port 37539
name server_node at port 45073
jose@uss-defiant:~$ 
```

Make sure epmd application is running.

- ps ax | grep epmd
- epmd &

Jinterface: receiving messages

Explicit cast operation

```
import com.ericsson.otp.erlang.*;

…

while (true) {

  OtpErlangObject newMessage = serverMailBox.receive();

  if (newMessage instanceof OtpErlangTuple){

      OtpErlangTuple erlangTuple = (OtpErlangTuple) newMessage;

      OtpErlangPid senderPID = (OtpErlangPid) erlangTuple.elementAt(0);

      OtpErlangString clientName = (OtpErlangString) erlangTuple.elementAt(1);

      String messageStr = "Hello " + clientName.stringValue();

      OtpErlangString greetings = new OtpErlangString(messageStr);

      serverMailBox.send(senderPID, greetings);

  }

}
```

Everytime a message is fetch, it is required to determine its type and perform some validations.

We make use of some classes to retrieve information sent by another process.

In this case I have the PID of the sender, I can make use of it to reply back.

We need to write code that is able to process incoming messages. We know their structure.

Getting ready our development environment (1)

Make sure you have installed:

1. Java SDK 11. (*)
2. Apache Maven 3.x version. (*)
3. Erlang OTP.
4. An IDE, it must support Maven.
5. A terminal (I use Terminator in Ubuntu).

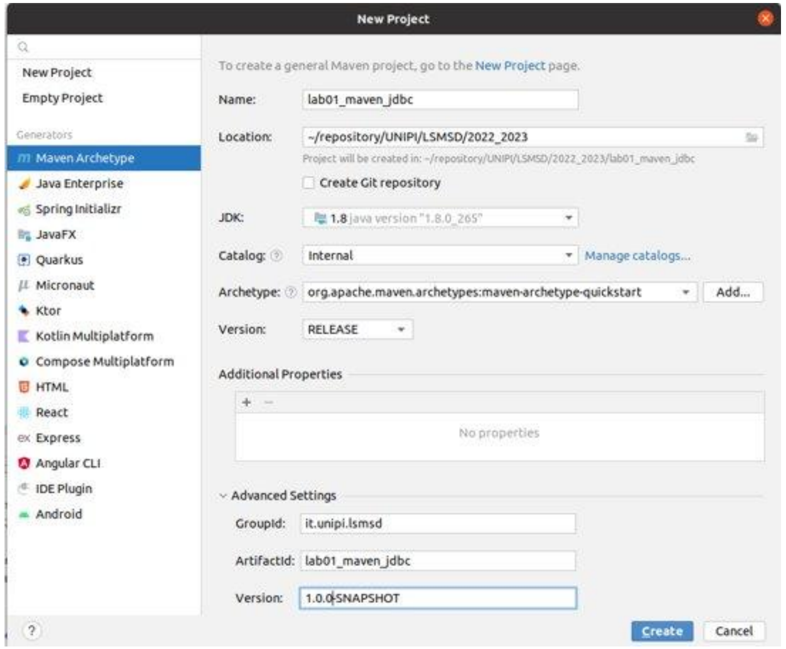
(*) Once you have installed the previous software packages, it is important to:

1. Define the **JDK_HOME** environment variable with the folder of your SDK 11 installation.
2. Define the **M2_HOME** environment variable with the folder of your Maven installation.
3. Add to the **PATH** environment variable the following paths (use $ in Linux % in Windows):
    a. **$JDK_HOME/bin**
    b. **$M2_HOME/bin**

(*) You can avoid doing all these step manually by installing SDKMAN: https://sdkman.io/

Getting ready our development environment (2)

Once you create your Maven project, DO NOT forget to delegate the build/actions to Maven (in IntelliJ).



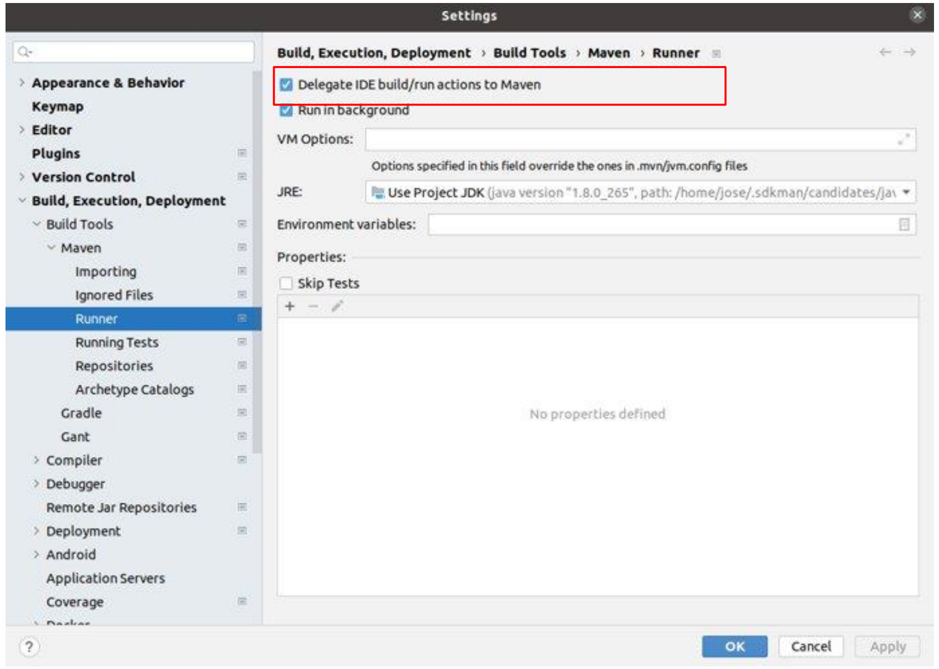
New Project

To create a general Maven project, go to the New Project page.

- Name: lab01_maven_jdbc
- Location: ~/repository/UNIPI/LSMSD/2022_2023
  Project will be created in: ~/repository/UNIPI/LSMSD/2022_2023/lab01_maven_jdbc
- [ ] Create Git repository
- JDK: 1.8 java version "1.8.0_265"
- Catalog: Internal — Manage catalogs...
- Archetype: org.apache.maven.archetypes:maven-archetype-quickstart — Add...
- Version: RELEASE

Additional Properties: No properties

Advanced Settings
- GroupId: it.unipi.lsmsd
- ArtifactId: lab01_maven_jdbc
- Version: 1.0.0-SNAPSHOT



Settings

Build, Execution, Deployment › Build Tools › Maven › Runner

- [✓] Delegate IDE build/run actions to Maven
- [✓] Run in background
- VM Options:
  Options specified in this field override the ones in .mvn/jvm.config files
- JRE: Use Project JDK (java version "1.8.0_265", path: /home/jose/.sdkman/candidates/ja
- Environment variables:
- Properties:
- [ ] Skip Tests

No properties defined

Exercise 1: Hello from Erlang!

Time: 25 min.

Using Maven, develop a Java console application that communicates with an Erlang process. **This application must:**

1. Ask a user their full name.
2. Send the message {PID, <full name entered in (1)>} to an Erlang process.
3. Read the reply by the Erlang process and print the greeting message. Format: {ok, <greeting message here>}.

From the Erlang process:

1. It receives a message with the following format {PID, <full name>}.
2. It builds a greeting message using the received full name.
3. It sends a response with the following format: {ok, <greeting message here>}.

Exercise 2: Fortune Cookies

Time: 30 min.

Using Maven, develop a Java console application that communicates with an Erlang process. **This application must:**

1. Ask a user his/her first and last name.
2. Send the message {PID, <first name entered in (1)>, <last name entered in (1)>} to an Erlang process.
3. Read the reply sent back by the Erlang process and print a fortune's cookie message. Format: {ok, <fortune's cookie message here>}.

From the Erlang process:

1. It receives a message with the following format {PID, <first name>, <last name>}.
2. It picks a random message from a list of fortune's cookie messages.
3. It builds a final message using the first and last name from (1) and the fortune's cookie message in (2).
4. It sends a response with the following format: {ok, <message built in (3)>}.

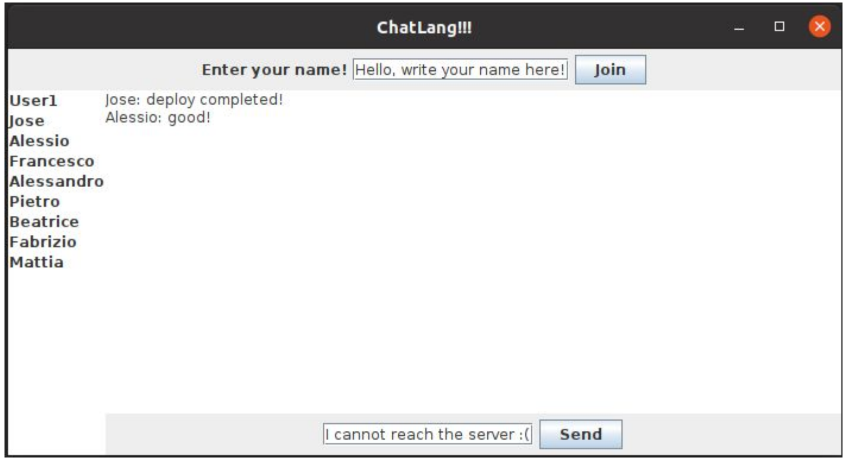
Bonus: you can consider loading these fortune's cookie messages from a text file.

Exercise 3: ChatLang (for home)

Time: 60 min.

Let’s create a Java Desktop application that allows users to participate in a chat. The application should have the shown UI and the its backend must be an Erlang application which handles the following functionalities:

- New users
- Broadcast new messages
- Users’ disconnections



Jose: deploy completed!
Alessio: good!

Exercise 4: Generating our executable (for home)

To run Exercise 2 and Exercises 3 Java applications, we make use of the IDE but…

What about if I would like to run them without using the IDE?

References

- https://maven.apache.org/guides/
- https://semver.org/
- https://netbeans.apache.org/tutorials/nbm-maven-quickstart.html
- https://www.infoworld.com/article/2071772/the-maven-2-pom-demystified.html?page=1
- https://www.oracle.com/technical-resources/articles/java/ma14-java-se-8-streams.html
- https://www.erlang.org/doc/apps/jinterface/jinterface_users_guide.html
- https://www.oracle.com/technical-resources/articles/linux/advanced-linux-command-mastery.html
- https://www.oracle.com/technical-resources/articles/linux/advanced-linux-command-mastery-part2.html

GitHub repo:

https://github.com/jlcorcuera/unipi-dsmt-2023-2024

Project's containers request form:

https://docs.google.com/forms/d/e/1FAIpQLSeD-5k7jSCETQ2nvb9yXPWcn19EmGg_pBY_tDReXdZHSLZwWA/viewform?usp=sharing