# Sandboxed Applications Training

# DISCLAIMER

All rights reserved. Information, technical data and tutorials contained in this document are proprietary under copyright law of MicroEJ S.A. Without written permission from MicroEJ S.A., copying or sending parts of the document or the entire document by any means to third parties is not permitted. Granted authorizations for using parts of the document or the entire document do not mean MicroEJ S.A. gives public full access rights.

The information contained herein is not warranted to be error-free.

MicroEJ® and all relative logos are trademarks or registered trademarks of MicroEJ S.A. in France and other Countries.

Other trademarks are proprietary of their respective owners.

Java™ is Sun Microsystems' trademark for a technology for developing application software and deploying it in cross-platform, networked environments. When it is used in this site without adding the "™" symbol, it includes implementations of the technology by companies other than Sun. Java™, all Java-based marks and all related logos are trademarks or registered trademarks of Sun Microsystems Inc, in the United States and other Countries.

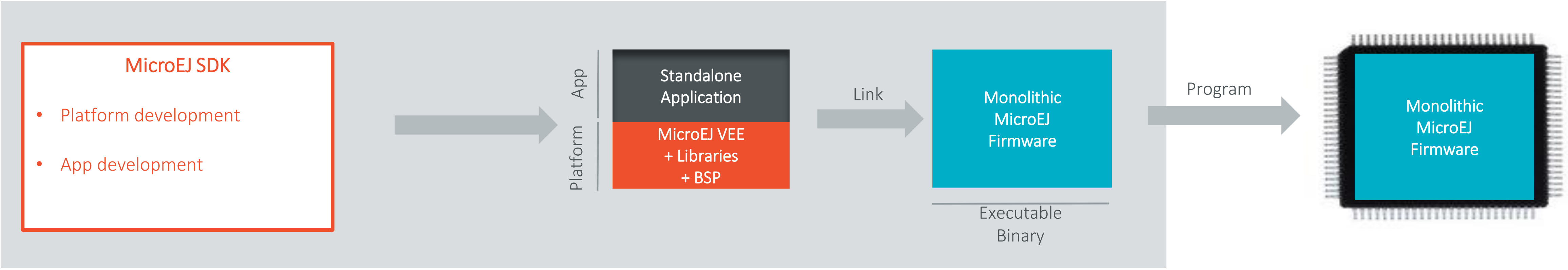
# INTRODUCTION

# AGENDA

- Sandboxed Applications Concepts.
- Sandboxed Application Project.
- Inter Application Communication: focus on Shared interface.

# Sandboxed Applications Concepts

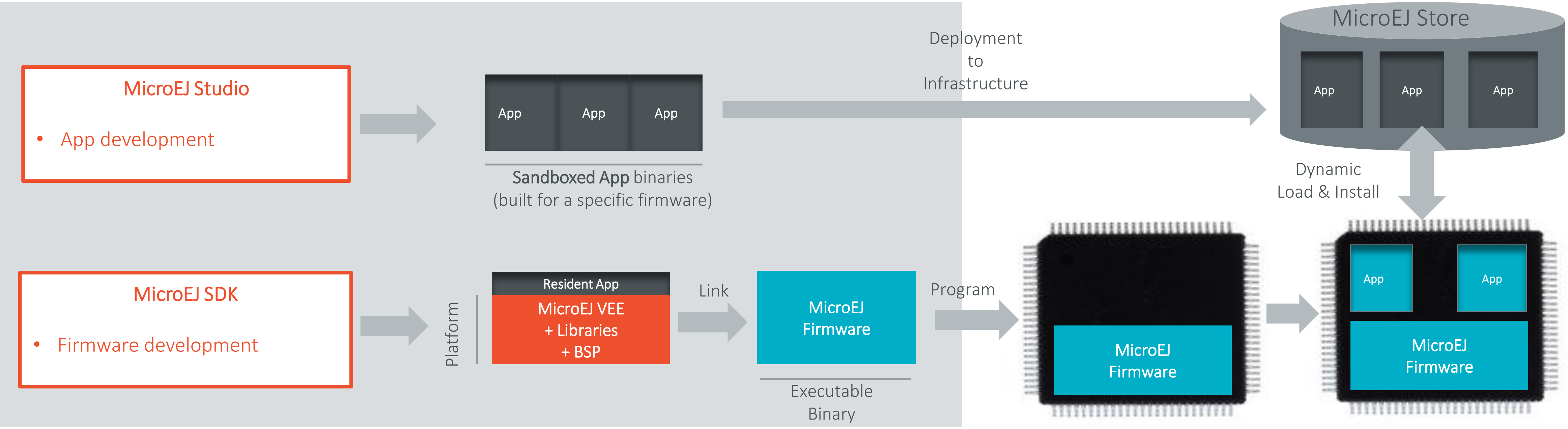
# MONO-SANDBOX DEVELOPMENT WORKFLOW



MicroEJ SDK

- Platform development
- App development

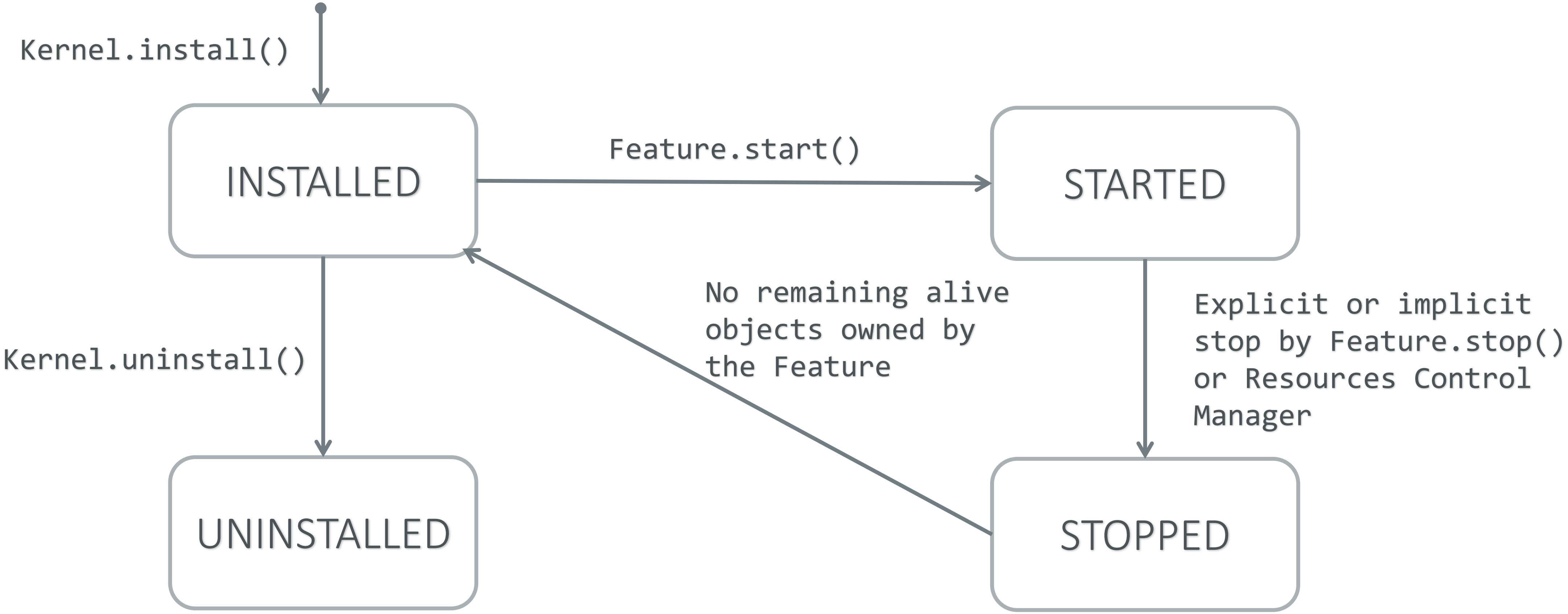
# MULTI-SANDBOX DEVELOPMENT WORKFLOW



# APPLICATION LIFECYCLE STATES (1/2)

- **INSTALLED:**
  - Application has been successfully linked to the Kernel and is not running. There are no references from the Kernel to objects owned by this application.
- **STARTED:**
  - Application has been started and is running.
- **STOPPED:**
  - Application has been stopped and all its threads are terminated. There are remaining references from the Kernel to objects owned by this Application.
- **UNINSTALLED:**
  - Application has been unlinked from the Kernel.

# APPLICATION LIFECYCLE STATES (2/2)



Kernel.install()

INSTALLED

Feature.start()

STARTED

Kernel.uninstall()

UNINSTALLED

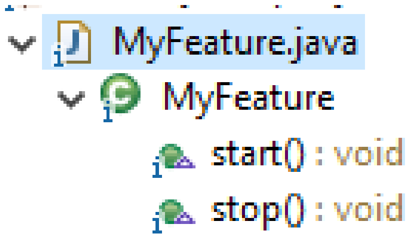
No remaining alive objects owned by the Feature

Explicit or implicit stop by Feature.stop() or Resources Control Manager

STOPPED

# ENTRY POINT

- A Sandboxed Application project needs to implement the **ej.kf.FeatureEntryPoint** interface.



MyFeature.java
MyFeature
start() : void
stop() : void

- **start()** is called after the application has been started:
  - Starts a thread or simply registers a shared interface.
- **stop()** is called just before the application is stopped:
  - Opportunity to save the state of the application (Properties)

# Sandboxed Application Project

Create and run a Sandboxed Application on a Kernel

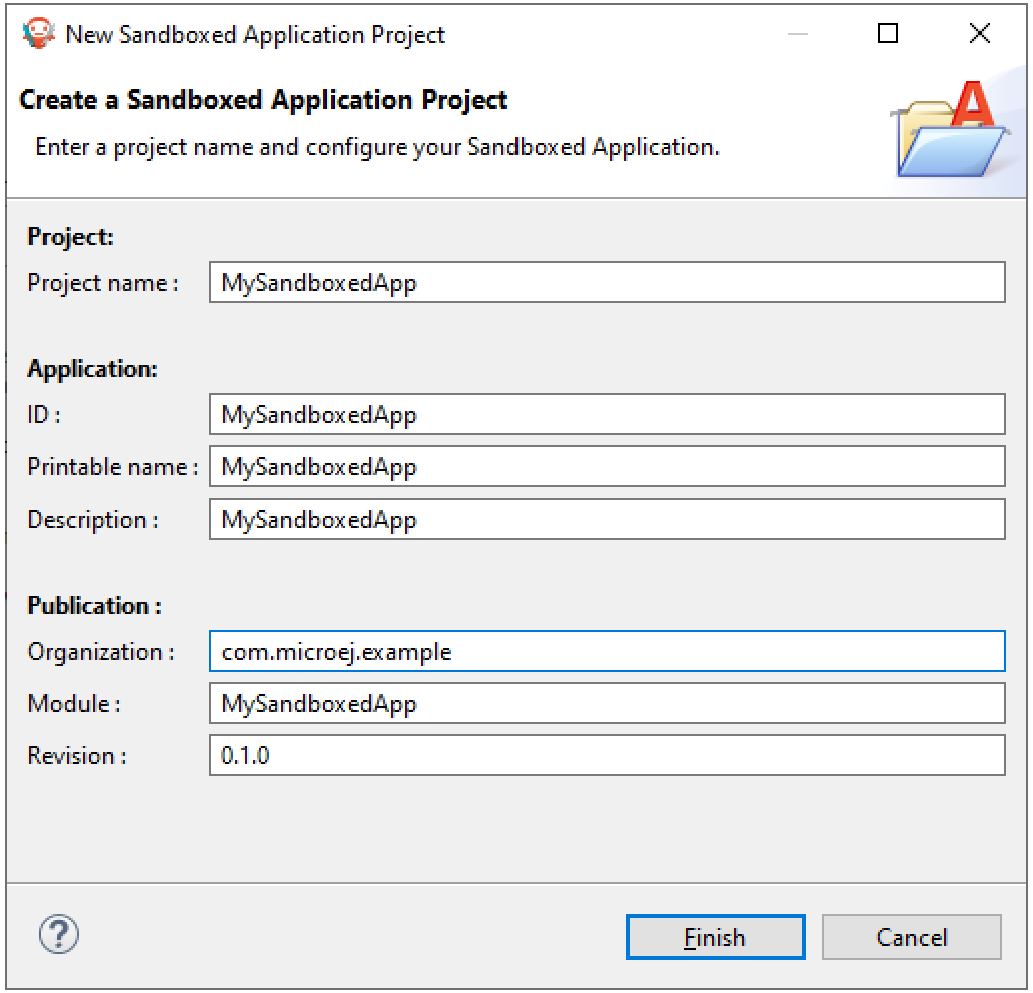
# PREREQUISITES

- Multi-Sandbox firmware.
- In this training, the STM32F7508-DK 1.2.0 Green firmware is used.
  - The list of firmware flavors can be found in the Kernel Developer Guide.
  - Download GREEN-STM32F7508-DK-1.2.0.out
  - Download GREEN-STM32F7508-DK-1.2.0.vde
- Import GREEN-STM32F7508-DK-1.2.0.vde in MICROEJ SDK:
  - Go to **File** -> **Import** -> **MicroEJ** -> **Virtual Device.**
  - Select the file.
  - Accept the license.
  - Click **Finish**.

- To get the lost of the imported Virtual Devices:
  - Go to **Window** -> **Preferences** -> **MicroEJ** -> **Virtual Devices.**

# CREATE THE APPLICATION PROJECT

- Go to **File** -> **New** -> **Sandboxed Application Project**.
- Fill the input fields.
- Click **Finish**.



**Create a Sandboxed Application Project**

Enter a project name and configure your Sandboxed Application.

**Project:**

Project name : MySandboxedApp

**Application:**

ID : MySandboxedApp

Printable name : MySandboxedApp

Description : MySandboxedApp

**Publication :**

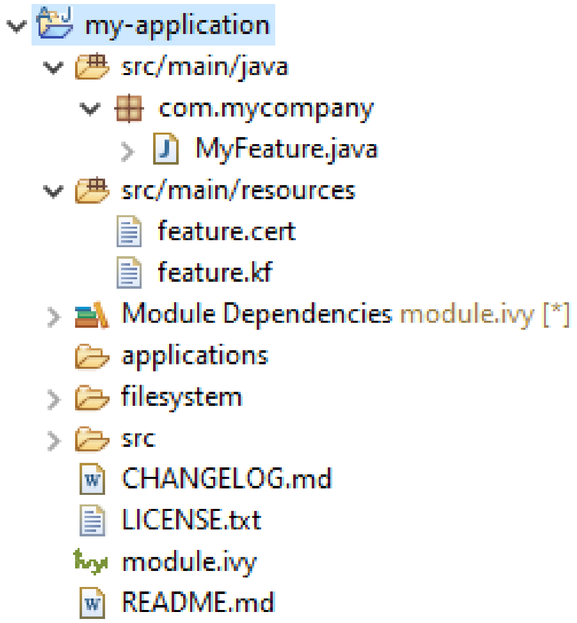
Organization : com.microej.example

Module : MySandboxedApp

Revision : 0.1.0

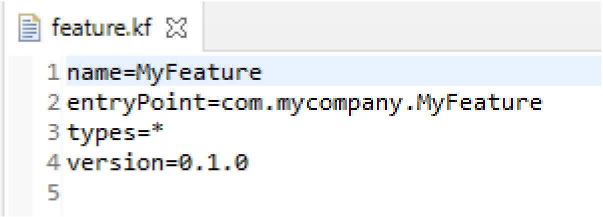
# PROJECT STRUCTURE

- **src/main/java:**
  - Sources folder.
- **src/main/resources:**
  - Folder for future resources (images, fonts, etc.).
  - **feature.cert**:
    - X509 certificate for identification (6 first fields RFC 2253)
  - **feature.kf**:
    - Defines the Kernel name and the version.
    - Defines the Application Entry Point.
    - List of all types that can be included in the Feature. Default value = “* “.
- **module.ivy:**
  - Module description file, dependencies description for the current project.



```
my-application
  src/main/java
    com.mycompany
      MyFeature.java
  src/main/resources
    feature.cert
    feature.kf
  Module Dependencies module.ivy [*]
  applications
  filesystem
  src
  CHANGELOG.md
  LICENSE.txt
  module.ivy
  README.md
```

Sandboxed Application project structure



feature.kf

```
name=MyFeature
entryPoint=com.mycompany.MyFeature
types=*
version=0.1.0

```

feature.kf file

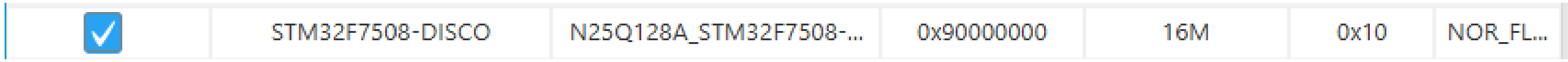
# RUN MY-APPLICATION IN SIM

- Right-Click on the **MyFeature.java** class
- Run As -> Run Configurations
- Double click on MicroEJ Application.
- Go to **Execution** tab:
  - Select the **VDE-Green** Virtual Device.
  - Select **Execute on Simulator.**
- Click **Run.**
- The Feature is started:

```
=============== [ Initialization Stage ] ===============
=============== [ Converting fonts ] ===============
=============== [ Converting images ] ===============
=============== [ Launching on Simulator ] ===============
=============== [ Launch Shielded Plug server on port 10082 ] ===============
ShieldedPlug client "/127.0.0.1:1991" disconnected.
ntpservice INFO: Start the ntp client
ntpservice INFO: Use the connectivity manager
commandserverentrypoint INFO: Start the admin server
Feature MyFeature started!
remotecommandserver INFO: Server listening on port 4000
ntpservice INFO: Scheduled update time task
ntpservice INFO: Update time Fri Jun 03 09:42:44 GMT 2024
ntpservice INFO: Stopped retry task
```

# RUN THE FIRMWARE ON DEVICE

- Connect your STM32F7058-DK board to your computer
- Install and open **STM32CubeProgrammer**
- Go to the EL section, select the following line:



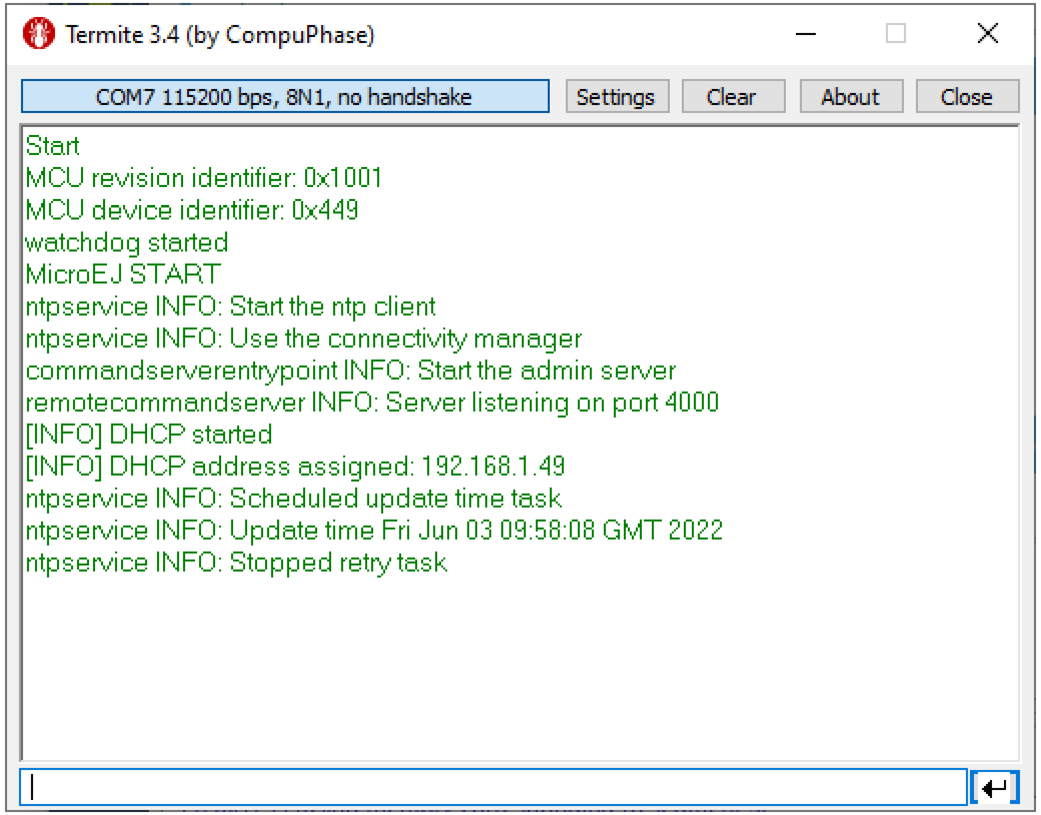
| | | | | | | |
|---|---|---|---|---|---|---|
| ☑ | STM32F7508-DISCO | N25Q128A_STM32F7508-... | 0x90000000 | 16M | 0x10 | NOR_FL... |

- Go to the ⬇ section, select the **GREEN-STM32F7508-DK-1.2.0.out** file.

- Click on the green button ‘Connect’ to connect **STM32CubeProgrammer** to your board.
- Once connected, click on **Start Programming** to program the board.

# GET THE FIRMWARE TRACES

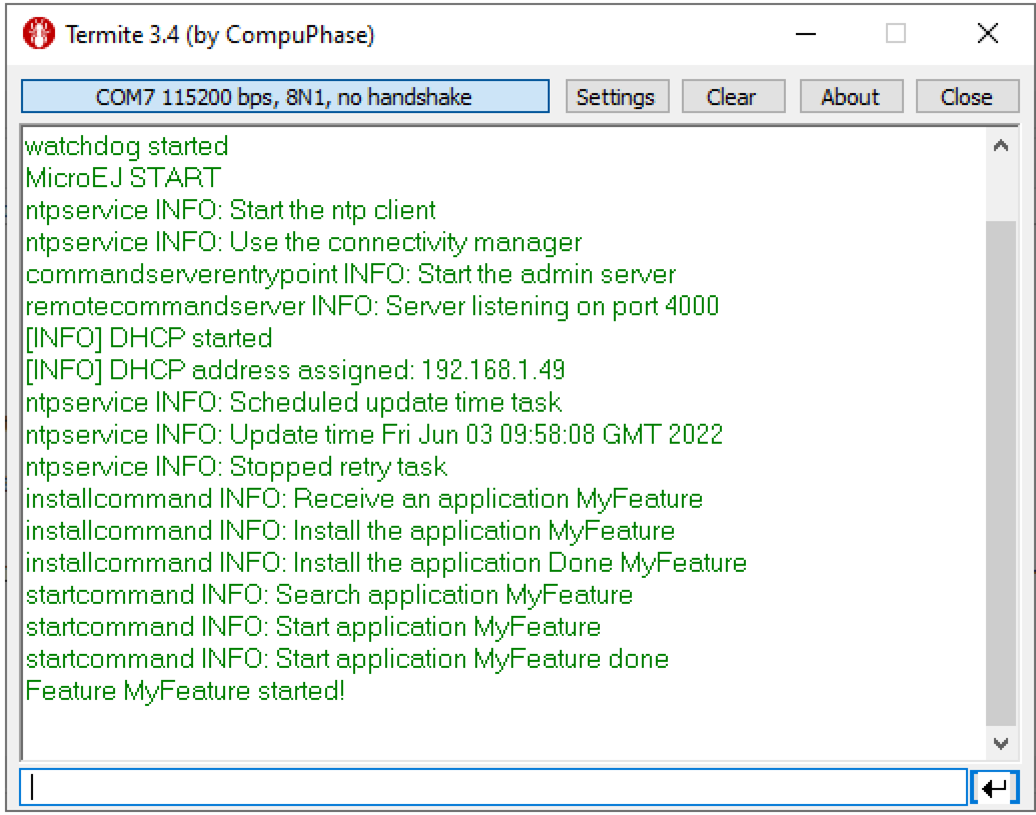
- Open the Termite serial terminal.
- Click the **Settings** button.
- Select the STM32F7508-DK board COM port.
- Reset the STM32F7508-DK board pressing the **black** button.
- The Kernel starts and logs are printed in the console.
- If the board is connected to the network, its local IP address is printed in the console.


Termite 3.4 (by CompuPhase)
COM7 115200 bps, 8N1, no handshake
Settings
Clear
About
Close

```
Start
MCU revision identifier: 0x1001
MCU device identifier: 0x449
watchdog started
MicroEJ START
ntpservice INFO: Start the ntp client
ntpservice INFO: Use the connectivity manager
commandserverentrypoint INFO: Start the admin server
remotecommandserver INFO: Server listening on port 4000
[INFO] DHCP started
[INFO] DHCP address assigned: 192.168.1.49
ntpservice INFO: Scheduled update time task
ntpservice INFO: Update time Fri Jun 03 09:58:08 GMT 2022
ntpservice INFO: Stopped retry task
```

# RUN MY-APPLICATION ON DEVICE

- Right-Click on the **MyFeature.java** class
- Run As -> Run Configurations
- Double click on MicroEJ Application.
- Go to **Execution** tab:
  - Select the **VDE-Green** Virtual Device.
  - Select **Execute on Device.**
  - Select the **Local Deployment (Socket)** option.
- Go to the **Configuration** tab:
  - In the **Local Deployment (Socket)** section, set the IP address of the board.
- Click **Run.**
- The Feature is installed and started on the device.

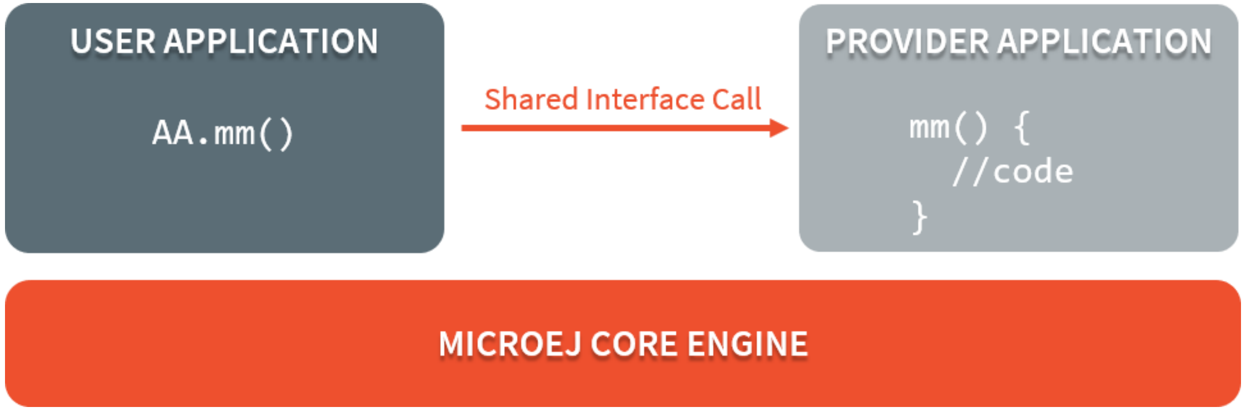

Termite 3.4 (by CompuPhase)
COM7 115200 bps, 8N1, no handshake
Settings
Clear
About
Close

```
watchdog started
MicroEJ START
ntpservice INFO: Start the ntp client
ntpservice INFO: Use the connectivity manager
commandserverentrypoint INFO: Start the admin server
remotecommandserver INFO: Server listening on port 4000
[INFO] DHCP started
[INFO] DHCP address assigned: 192.168.1.49
ntpservice INFO: Scheduled update time task
ntpservice INFO: Update time Fri Jun 03 09:58:08 GMT 2022
ntpservice INFO: Stopped retry task
installcommand INFO: Receive an application MyFeature
installcommand INFO: Install the application MyFeature
installcommand INFO: Install the application Done MyFeature
startcommand INFO: Search application MyFeature
startcommand INFO: Start application MyFeature
startcommand INFO: Start application MyFeature done
Feature MyFeature started!
```

# Shared Interfaces

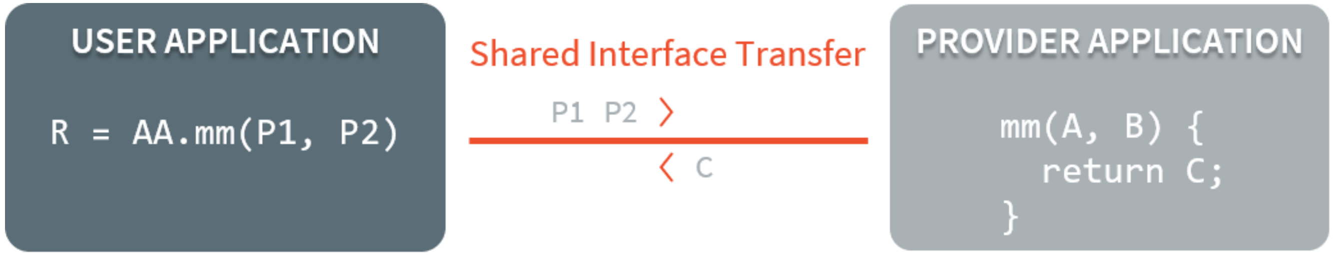
# OVERVIEW

- The Shared Interface mechanism provided by MicroEJ Core Engine is an object communication bus based on plain Java interfaces where method calls are allowed to cross MicroEJ Sandboxed Applications boundaries.
- The Shared Interface mechanism is the cornerstone for designing reliable Service Oriented Architectures on top of MicroEJ. Communication is based on the sharing of interfaces defining APIs (Contract Oriented Programming).
- The basic schema:
  - A provider application publishes an implementation for a shared interface into a system registry.
  - A user application retrieves the implementation from the system registry and directly calls the methods defined by the shared interface.



# TRANSFERABLE TYPES (1/3)

- In the process of a cross-application method call, parameters and return value of methods declared in a Shared Interface must be transferred back and forth between application boundaries.



- Some restrictions apply to Shared Interfaces compared to standard java interfaces:
  - Types for parameters and return values must be **transferable types**.
  - Thrown exceptions must be classes owned by the MicroEJ Firmware.

# TRANSFERABLE TYPES (2/3)

- The table bellow describes the rules applied depending on the element to be transferred:

| Type | Type Owner | Instance Owner | Rule |
|---|---|---|---|
| Primitive Type | N/A | N/A | Passing by value. (boolean, byte, short, char, int, long, double, float) |
| Any Class, Array or Interface | Kernel | Kernel | Passing by reference |
| Any Class, Array or Interface | Kernel | Application | MicroEJ Kernel specific or forbidden |
| Array of base types | Any | Application | Clone by copy |
| Arrays of references | Any | Application | Clone and transfer rules applied again on each element |
| Shared Interface | Application | Application | Passing by indirect reference (Proxy creation) |
| Any Class, Array or Interface | Application | Application | Forbidden |

# TRANSFERABLE TYPES (3/3)

- Objects created by a Sandboxed Application which type is owned by the Kernel can be transferred to another Sandboxed Application provided this has been authorized by the Kernel.

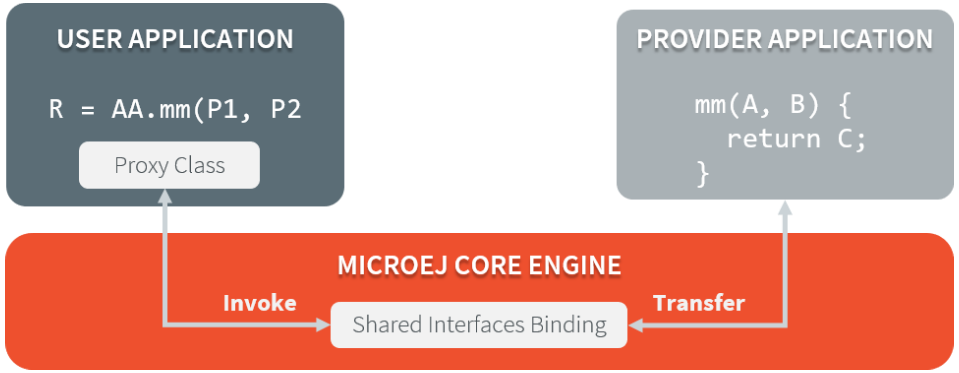
- The list of Kernel types that can be transferred is Kernel specific, so you have to consult your Kernel specification.

- When an argument transfer is forbidden, the call is abruptly stopped and a **java.lang.IllegalAccessError** is thrown by the Core Engine.

- For the forbidden types to be transferable, a dedicated Kernel Type Converter must have been registered in the Kernel.

# PROXY CLASS (1/2)

- The Shared Interface mechanism is based on automatic proxy objects created by the underlying MicroEJ Core Engine, so that each application can still be dynamically stopped and uninstalled.
- This offers a reliable way for users and providers to handle the relationship in case of a broken link.
- Once a Java interface has been declared as Shared Interface, a dedicated implementation is required (called the Proxy class implementation).



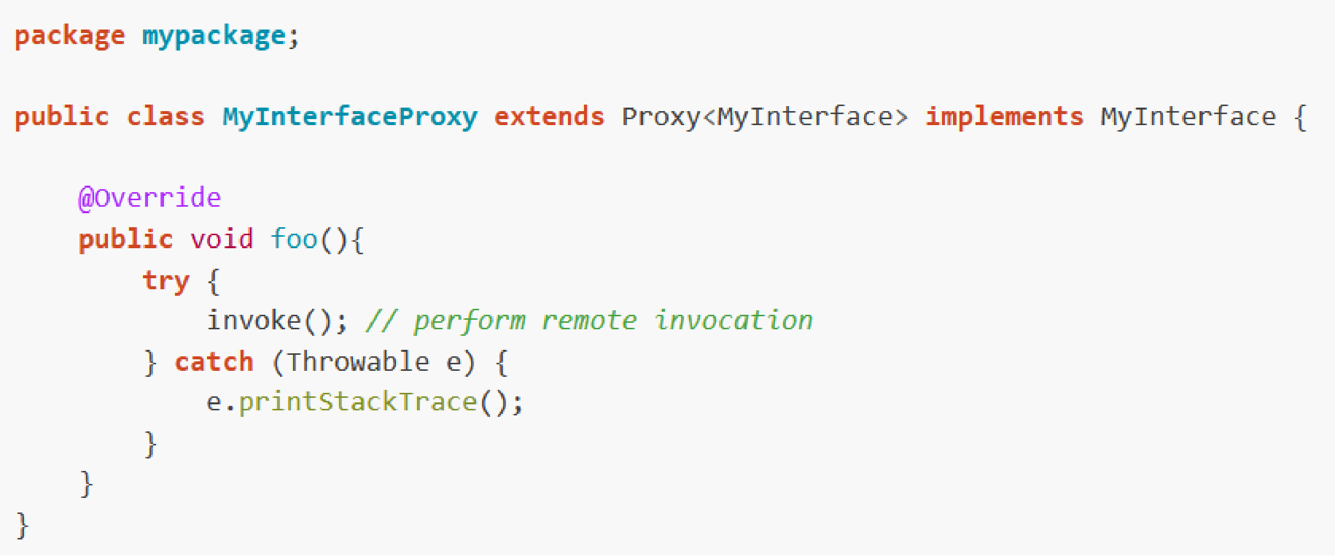
```
R = AA.mm(P1, P2
```

```
mm(A, B) {
  return C;
}
```

- Its main goal is to perform the remote invocation and provide a reliable implementation regarding the interface contract even if the remote application fails to fulfill its contract (unexpected exceptions, application killed…).
- The MicroEJ Core Engine will allocate instances of this class when an implementation owned by another application is being transferred to this application.

# PROXY CLASS (2/2)

- A proxy class is implemented and executed on the client side, each method of the implemented interface must be defined according to the following pattern:



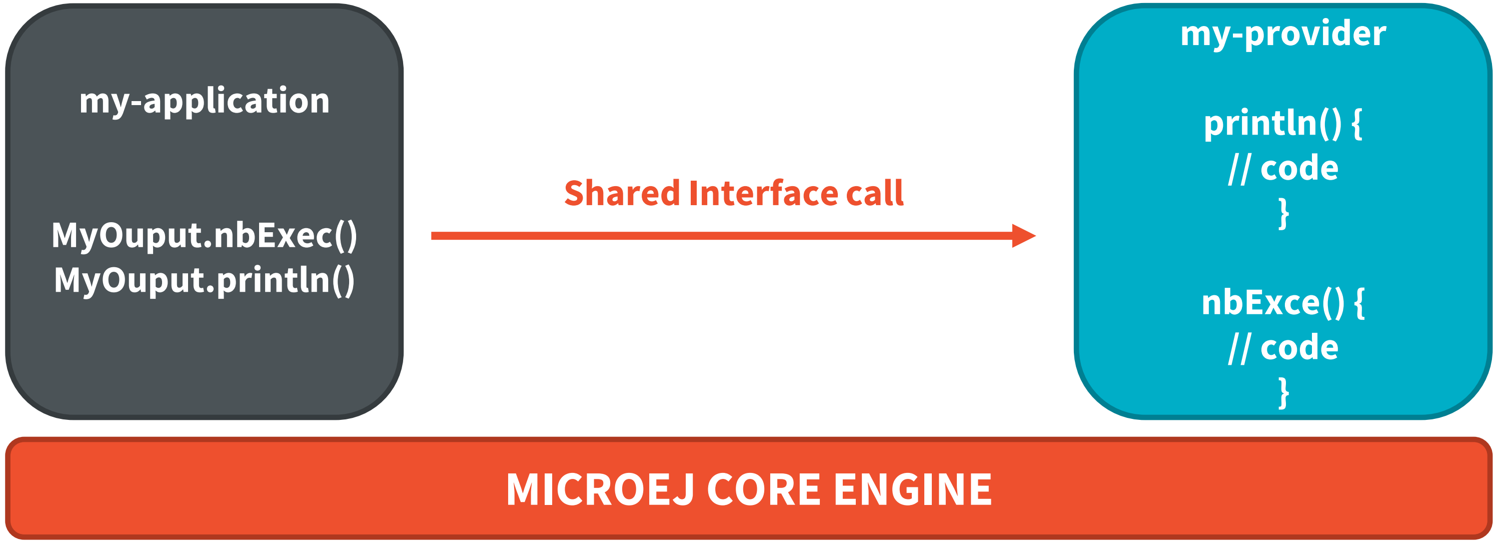
```java
package mypackage;

public class MyInterfaceProxy extends Proxy<MyInterface> implements MyInterface {

    @Override
    public void foo(){
        try {
            invoke(); // perform remote invocation
        } catch (Throwable e) {
            e.printStackTrace();
        }
    }
}
```

- Each implemented method of the proxy class is responsible for performing the remote call and catching all errors from the server side and to provide an appropriate answer to the client application call according to the interface method specification (contract).
- Remote invocation methods are defined in the super class **ej.kf.Proxy** and are named **invokeXXX()** where **XXX** is the kind of return type.

# Hand’s On

# HAND’S ON OVERVIEW

- A **my-provider** application provides a **MyOutput** service with two methods (**println** / **nbExec**).
- **my-application** will be updated to call this methods using the shared interface mechanism.



# Update my-application

Update the application to call the MyOutput service provided by  the provider application.

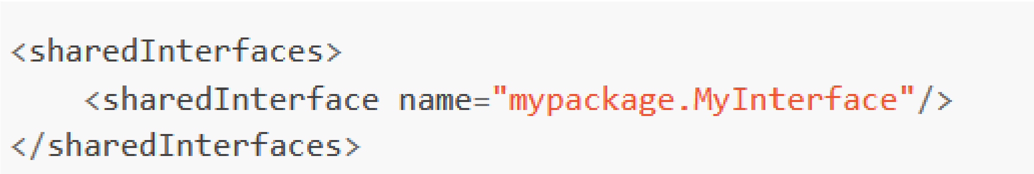
# INTERFACE DEFINITION (1/2)

- The definition of a Shared Interface starts by defining a standard Java interface.
- In the **my-application** project:
  - Create a new package **com.microej.example.sharedinterface.shared**
  - Create a **MyOutput** Interface :

```
package com.microej.example.sharedinterface.shared;

import java.io.IOException;

public interface MyOutput {
/**
 * Print function.
 *
 * @param str
 *           The string to print.
 * @throws IOException
 *            Throws an IOException when the service is not available.
 */
void println(String str) throws IOException;
/**
 * Returns the number of time the println has been executed.
 *
 * @return the number of time the println has been executed.
 */
int nbExec();
}
```

# INTERFACE DEFINITION (2/2)

- To declare an interface as a Shared Interface, it must be registered in a Shared Interfaces identification file.
- A Shared Interface identification file is an XML file with the **.si** suffix with the following format:



```
<sharedInterfaces>
    <sharedInterface name="mypackage.MyInterface"/>
</sharedInterfaces>
```

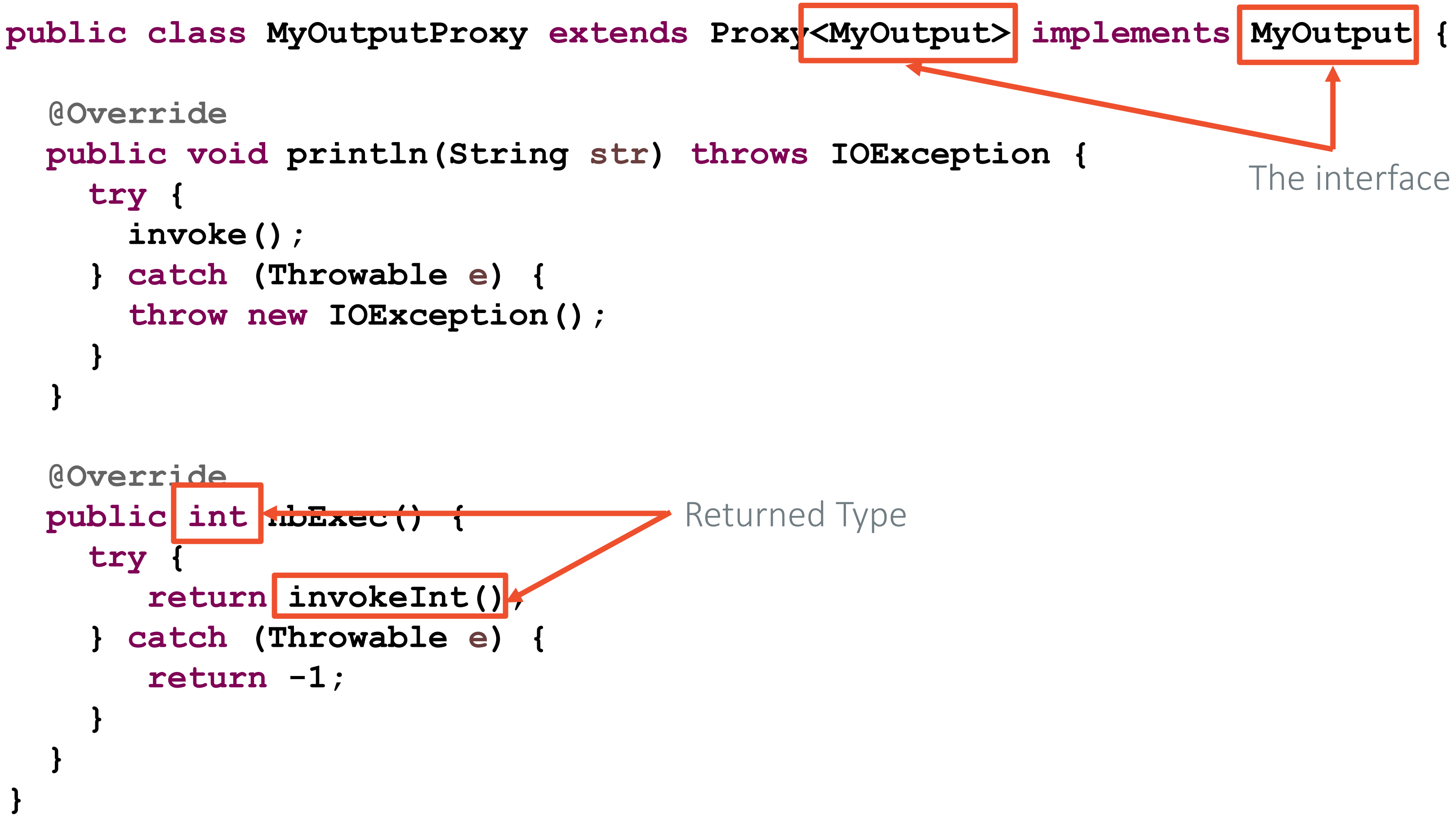
- Shared Interface identification files must be placed at the root of a path of the application classpath.
- For a MicroEJ Sandboxed Application project, it is typically placed in **src/main/resources** folder.

- **Hand's on:**
    - Add a **sharedInterfaces.si** file in the **src/main/resources** folder of the Application project:

```
<sharedInterfaces>
        <sharedInterface name="com.microej.example.sharedinterface.shared.MyOutput" />
</sharedInterfaces>
```

# PROXY IMPLEMENTATION

- In the **my-application** project:
  - Create a sub-class of **Proxy** that implements **MyOutput**:
  - **Note: the Proxy class name must follow the following pattern: {InterfaceName}Proxy and should be put in the same package that the interface.**



```
public class MyOutputProxy extends Proxy<MyOutput> implements MyOutput {

  @Override
  public void println(String str) throws IOException {
    try {
      invoke();
    } catch (Throwable e) {
      throw new IOException();
    }
  }

  @Override
  public int nbExec() {
    try {
      return invokeInt();
    } catch (Throwable e) {
      return -1;
    }
  }
}
```

The interface

Returned Type

# USE THE SERVICE

- In the **my-application** project:
  - In the **module.ivy**, add the following dependency:

```
<dependency org="ej.library.runtime" name="service" rev="1.1.1" />
```

  - Update the application **start()** method code to use a Timer task that periodically uses the service:

```
ej.bon.Timer myTimer = new ej.bon.Timer();
myTimer.schedule(new ej.bon.TimerTask() {

    @Override
    public void run() {
        MyOutput output = ServiceFactory.getService(MyOutput.class);
        if (output != null) {
            try {
                output.println("Hello World ! n° " + output.nbExec());
            } catch (IOException e) {
                System.out.println("MyOutput Service unavailable !");
            }
        }
    }
}, 0, 1000);
```

# my-provider

Implement **MyOutput** in an other application

# IMPORT THE MY-PROVIDER APPLICATION

- Import the **my-provider** application template:
  - Open menu **File** > **Import…** > **General** > **Existing Projects into Workspace.**
  - Select the archive file **[training-package]/my-provider.zip**
  - Click on Finish.
- The **my-provider** application template is equivalent to the **my-application** template. The following elements have been modified:
  - Entry Point class has been renamed.
  - Print message has been updated to distinguish the 2 applications.
  - **kernel.kf** has been updated to change the Feature name and Entry Point.

# IMPLEMENT THE SERVICE

- In **my-provider**:
  - Copy the following elements from **my-application** to **my-provider**:
    - **MyOutput**, **MyOutputProxy, sharedInterface.si**.
  - Create a new class **MyStandardOutput** that implements **MyOutput**:

```java
package com.microej.example.sharedinterface.shared;

import java.io.IOException;

public class MyStandardOutput implements MyOutput {

    private int nbExec = 0;

    @Override
    public void println(String str) throws IOException {
        this.nbExec++;
        System.out.println("MyOutput Print : " + str);
    }

    @Override
    public int nbExec() {
        return this.nbExec;
    }
}
```

# REGISTER THE SERVICE

- In **my-provider**:
  - Add the following dependency to **module.ivy**:

```
<dependency org="ej.library.wadapps" name="wadapps" rev="2.1.1" />
```

  - Update the **MyFeatureProvider** class.

```
public class MyFeatureProvider implements FeatureEntryPoint {

private final MyStandardOutput standardOutput = new MyStandardOutput();

    @Override
    public void start() {
        System.out.println("Feature MyFeatureProvider started!"); //$NON-NLS-1$
        SharedServiceFactory.getSharedServiceRegistry().register(MyOutput.class, this.standardOutput);
    }

    @Override
    public void stop() {
        System.out.println("Feature MyFeatureProvider stopped!"); //$NON-NLS-1$
        SharedServiceFactory.getSharedServiceRegistry().unregister(MyOutput.class, this.standardOutput);
    }
}
```

# RUN THE EXAMPLE IN SIM

- The **my-provider** application should be built in order to be used by **my-application**:
  - Right-Click on **my-provider.**
  - Click **Build Module.**
  - A **target~/** folder appears in the project folder.

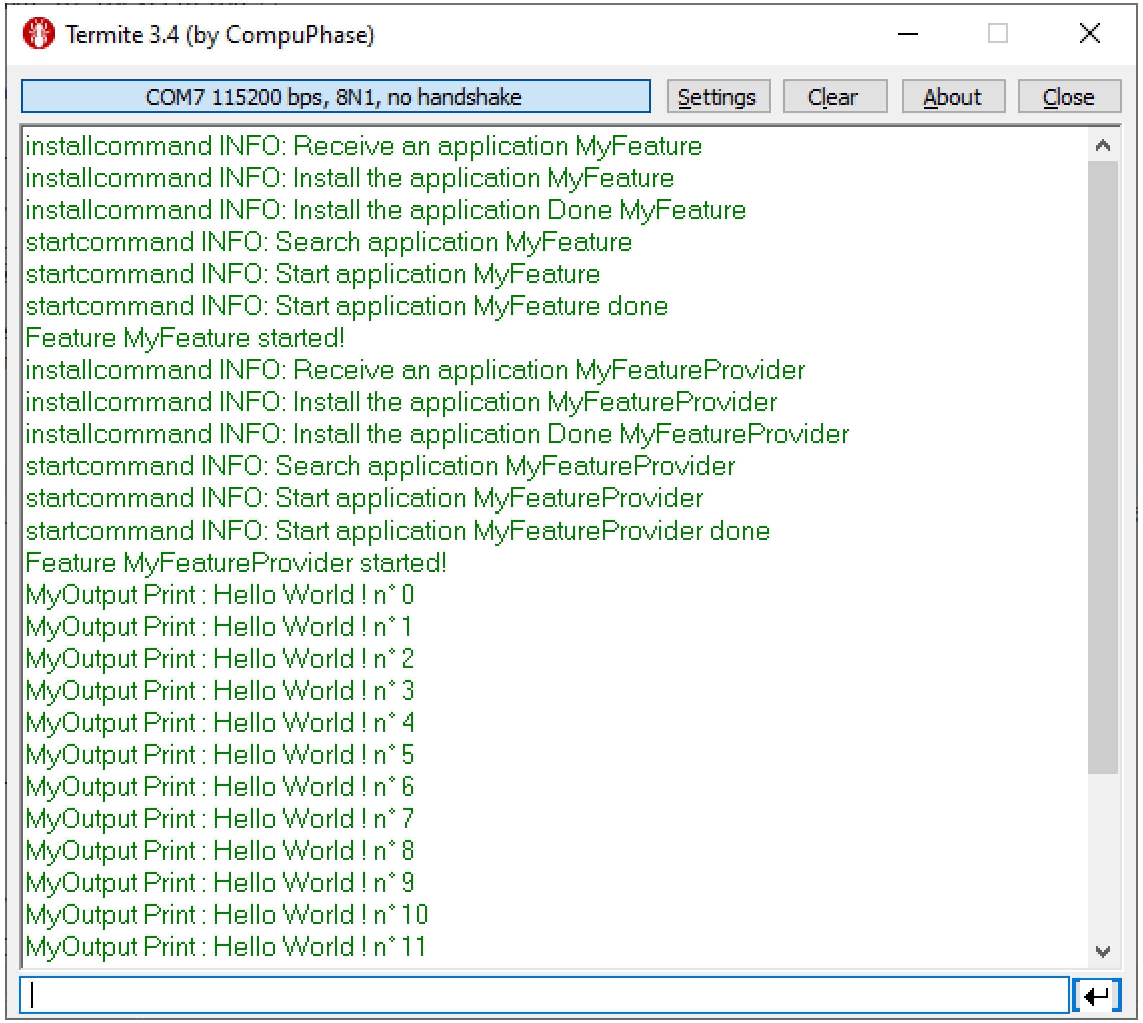
- Open the **my-application SIM** launcher.
- Go to the **Configuration** tab:
  - In the **Simulator -> Applications** section, set the path to generated artifacts of my-provider: **${project_loc:my-provider}/target~/artifacts.**
- Click **Run.**
- The Features are started. **my-application** uses the service provided by **my-provider.**

```
=============== [ Initialization Stage ] ===============
=============== [ Converting fonts ] ===============
=============== [ Converting images ] ===============
=============== [ Launching on Simulator ] ===============
=============== [ Launch Shielded Plug server on port 10082 ] ===============
ShieldedPlug client "/127.0.0.1:1447" disconnected.
Feature MyFeatureProvider started!
Feature MyFeature started!
ntpservice INFO: Start the ntp client
ntpservice INFO: Use the connectivity manager
MyOutput Print : Hello World ! N° 0
commandserverentrypoint INFO: Start the admin server
remotecommandserver INFO: Server listening on port 4000
MyOutput Print : Hello World ! N° 1
ntpservice INFO: Scheduled update time task
ntpservice INFO: Update time Fri Jun 03 13:33:52 GMT 2024
ntpservice INFO: Stopped retry task
MyOutput Print : Hello World ! N° 2
MyOutput Print : Hello World ! N° 3
```

# RUN THE EXAMPLE ON DEVICE

- Deploy **my-application** and **my-provider** on the device.

- Open the Termite serial terminal.
- Click the **Settings** button.
- Select the STM32F7508-DK board COM port.
- Reset the STM32F7508-DK board pressing the **black** button.
- The Features are installed and started. **my-application** uses the service provided by **my-provider.**


Termite 3.4 (by CompuPhase)
COM7 115200 bps, 8N1, no handshake
Settings
Clear
About
Close

```
installcommand INFO: Receive an application MyFeature
installcommand INFO: Install the application MyFeature
installcommand INFO: Install the application Done MyFeature
startcommand INFO: Search application MyFeature
startcommand INFO: Start application MyFeature
startcommand INFO: Start application MyFeature done
Feature MyFeature started!
installcommand INFO: Receive an application MyFeatureProvider
installcommand INFO: Install the application MyFeatureProvider
installcommand INFO: Install the application Done MyFeatureProvider
startcommand INFO: Search application MyFeatureProvider
startcommand INFO: Start application MyFeatureProvider
startcommand INFO: Start application MyFeatureProvider done
Feature MyFeatureProvider started!
MyOutput Print : Hello World ! n° 0
MyOutput Print : Hello World ! n° 1
MyOutput Print : Hello World ! n° 2
MyOutput Print : Hello World ! n° 3
MyOutput Print : Hello World ! n° 4
MyOutput Print : Hello World ! n° 5
MyOutput Print : Hello World ! n° 6
MyOutput Print : Hello World ! n° 7
MyOutput Print : Hello World ! n° 8
MyOutput Print : Hello World ! n° 9
MyOutput Print : Hello World ! n° 10
MyOutput Print : Hello World ! n° 11
```

# Tools

# ADMIN CONSOLE

- Go to **Run -> Run Configuration.**
- Double-click **MicroEJ Tool**.
- Go to **Execution** tab:
    - Select the **VDE-Green** Virtual Device.
    - Select the tool: **Wadapps Admin Console over Socket.**
- Click **Run.**

```
=============== [ Connect to 192.168.111.3:4000 ] ===============
*** Admin console ready ***
$
```

- Click the red square to stop the Admin Console.

# ADMIN CONSOLE COMMANDS

- Main commands:
    - help
    - man
    - list
    - start
    - stop
    - install
    - uninstall
    - exit

# THANK YOU

*for your attention !*

MICROEJ®