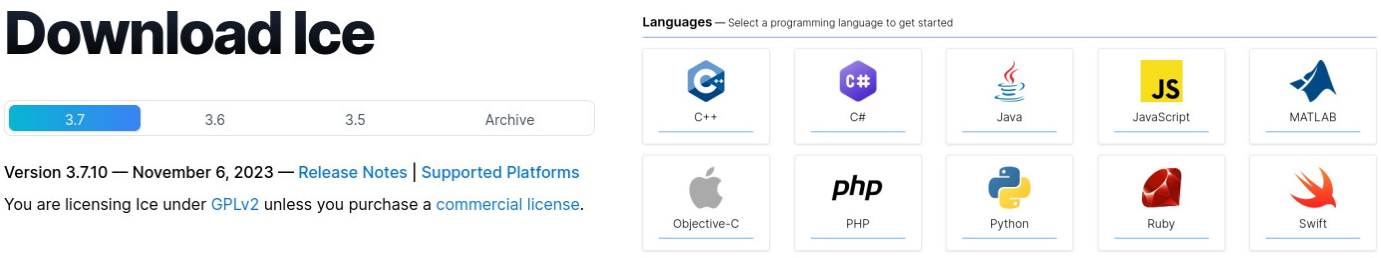
Hello World Application
ZeroC ICE

Requisitos:
jdk 11
ZeroICE 3.7
gradle 8.6

Instalar ZeroICE: https://zeroc.com/ice/downloads/3.7
descargar ICE para Java



Seleccionar su sistema operativo y descargar el instalador:

Realizar un proyecto
en este caso llamaremos el proyecto myapp. Usted puede hacerlo con el nombre sugerido en la pagina:
printer
Seguiremos el tutorial en la pagina:
https://doc.zeroc.com/ice/latest/hello-world-application/writing-an-ice-application-with-java

1. Crear la carpeta del proyecto e inicializar usando gradle
mkdir myapp
cd myapp
gradle init

en gradle usamos las opciones:
basic
groovy

default
default

2. con el proyecto inicializado, vamos a editar el archivo build.gradle con el siguiente contenido. puedes borrar los comentarios iniciales

```
plugins {
    id 'com.zeroc.gradle.ice-builder.slice' version '1.5.0' apply false
}

subprojects {
    //
    // Apply Java and Ice Builder plug-ins to all sub-projects
    //
    apply plugin: 'java'
    apply plugin: 'com.zeroc.gradle.ice-builder.slice'

    //
    // Both Client and Server projects share the Printer.ice Slice definitions
    //
    slice {
        java {
            files = [file("../Myapp.ice")]
        }
    }

    //
    // Use Ice JAR files from maven central repository
    //
    repositories {
        mavenCentral()
    }

    //
    // Both Client and Server depend only on Ice JAR
    //
    dependencies {
        implementation 'com.zeroc:ice:3.7.2'
    }

    //
    // Create a JAR file with the appropriate Main-Class and Class-Path attributes
    //
    jar {
        manifest {
            attributes(
               "Main-Class": project.name.capitalize(),
               "Class-Path": configurations.runtimeClasspath.resolve().collect { it.toURI() }.join(' ')
            )
        }
    }
}
```

3. editar el archivo settings.gradle

```
rootProject.name = 'myapp'
include 'client'
include 'server'
```

4. Crear las carpetas para cliente y servidor
mkdir client

mkdir server

5. El siguiente paso es agregar un archivo Slice (Specification Language for Ice).
Cuando se construya el proyecto, el plugin de ICE compila el archivo slice. Lo que debes hacer es crear el archivo **Myapp.ice** con el siguiente contenido

```
module Demo
{
    interface Printer
    {
        void printString(string s);
    }
}
```

6. Crear el servidor en java: mkdir server/src/main/java/
Debemos crear la clase **MyappI.java** que implementa la interface **printer**
con el siguiente código fuente

```
public class MyappI implements Demo.Printer
{
    public void printString(String s, com.zeroc.Ice.Current current)
    {
        System.out.println(s);
    }
}
```

7. Crear la clase **Server.java** en server/src/main/java/Server.java

```
public class Server
{
    public static void main(String[] args)
    {
        try(com.zeroc.Ice.Communicator communicator = com.zeroc.Ice.Util.initialize(args))
        {
            com.zeroc.Ice.ObjectAdapter adapter =
communicator.createObjectAdapterWithEndpoints("SimplePrinterAdapter", "default -p 10000");
            com.zeroc.Ice.Object object = new MyappI();
            adapter.add(object, com.zeroc.Ice.Util.stringToIdentity("SimplePrinter"));
            adapter.activate();
            communicator.waitForShutdown();
        }
    }
}
```

8. Construir el proyecto:
.\gradlew build

Compilar el servidor
.\gradlew :server:build

9. Crear el **cliente** en java. client/src/main/java/Client.java

```
public class Client
{
    public static void main(String[] args)
    {
        try(com.zeroc.Ice.Communicator communicator = com.zeroc.Ice.Util.initialize(args))
        {
           com.zeroc.Ice.ObjectPrx base = communicator.stringToProxy("SimplePrinter:default -p 10000");
            Demo.PrinterPrx printer = Demo.PrinterPrx.checkedCast(base);
            if(printer == null)
            {
                throw new Error("Invalid proxy");
            }
            printer.printString("Hola Mundo!!");
        }
    }
}
```

10. compilar el cliente
.\gradlew :client:build

11. ejecutar cliente y servidor. Esto se debe hacer en consolas separadas
servidor
java -jar server/build/libs/server.jar

cliente
java -jar client/build/libs/client.jar

Si todo sale bien, el servidor debe imprimir "Hola Mundo!!"

Ejercicios

Modifica el cliente para que acepte una cadena por línea de comandos en lugar de enviar siempre "Hello World". Comprueba que funciona.

Ejecuta varios clientes que envían cadenas distintas.

Con la herramienta más simple que conozcas, comprueba que el servidor efectivamente está escuchando en el puerto especificado.

Modifica el sirviente para que acepte una cadena de texto en el constructor y la imprima junto con cada mensaje entrante.

Modifica el servidor creando varios sirvientes MyappI y añadiéndolos al adaptador PrinterAdapter. Imprime los proxies de cada uno de ellos.

Invócalos desde distintos clientes. Ahora escribe un cliente que crea varios objetos, pero que utiliza la misma instancia del sirviente para todos ellos.

Intenta ejecutar el servidor en dos terminales distintos en el mismo computador ¿qué ocurre?