Ice: Hello world

1 printer.ice

```
module Example {
  interface Printer {
    void write(string message);
  };
};
```

Especificación ***SLICE*** para un "*Hola mundo*" distribuido

2

```
$ slice2java Printer.ice
```

Los **stubs** incluyen una versión básica de la interfaz Printer en el lenguaje de programación elegido. Por ejemplo Java

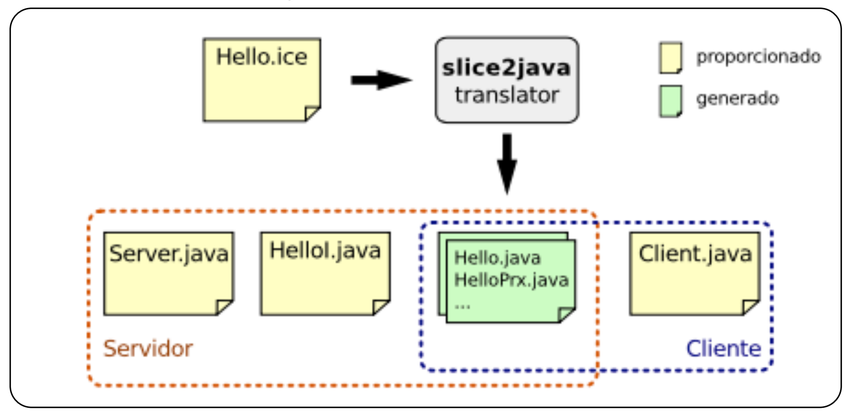
stubs
para el cliente y el servidor

slice2java genera (entre otras) la interfaz **_PrinterDisp** de **Printer** para el lenguaje Java

```
public abstract class _PrinterDisp extends Ice.ObjectImpl implements Printer
{
    protected void
    ice_copyStateFrom(Ice.Object __obj)
        throws java.lang.CloneNotSupportedException
    {
        throw new java.lang.CloneNotSupportedException();
    }

    public static final String[] __ids =
    {
        "::Example::Printer",
        "::Ice::Object"
    };
```

Esquema de general de compilación de Cliente y Servidor en Java



Example
- Printer.ice
- _PrinterDisp.java
- _PrinterOperations.java
- _PrinterOperationsNC.java
- Callback_Printer_write.java
- Printer.java
- PrinterHolder.java
- PrinterI.java
- PrinterPrx.java
- PrinterPrxHelper.java
- PrinterPrxHolder.java

3

```
$ slice2java --impl Printer.ice
```

Una implementación básica del sirviente se puede generar automáticamente con la opción *-impl* de *slice2java*

El sirviente **PrinterI hereda** de la interfaz **_PrinterDisp**

```
public final class PrinterI extends _PrinterDisp {
    public
    PrinterI() {

    }

    public void
    write(String message, Ice.Current __current) {
        System.out.println(message);
    }
}
```

4

PrinterI es la Implementación del ***Sirviente*** de la aplicación **Printer** que hereda de la interfaz **_PrinterDisp**

El sirviente **PrinterI** se *instancia* en el **Servidor**

5

Server : El Servidor consiste en implementar una clase que hereda de ***Ice.Applicaction*** en donde sólo hay que implementar el método ***run()***

```
public class Server extends Ice.Application {
    public int run(String[] args) {

        Ice.Object servant = new PrinterI();

        ObjectAdapter adapter = communicator().createObjectAdapter("PrinterAdapter");

        ObjectPrx proxy = adapter.add(servant, Util.stringToIdentity("printer1"));

        System.out.println(communicator().proxyToString(proxy));

        adapter.activate();

        shutdownOnInterrupt();

        communicator().waitForShutdown();

        return 0;
    }

    static public void main(String[] args) {
        Server app = new Server();
        app.main("Server", args);
    }
}
```

Se crea el ***sirviente***

El **communicator** representa el **broker** de objetos del **núcleo** de **comunicaciones**

El **adaptador** requiere un ***endpoint*** especificado por un protocolo de conexión a la red y definido en un **archivo de configuración** (Server.config)

Se crea el **adaptador de objetos** encargado de multiplicar entre los objetos almacenados en le servidor

```
PrinterAdapter.Endpoints=tcp -p 9090
```

Activación del adaptador que se ejecuta en otro hilo. A partir de este momento el **servidor puede escuchar y procesar peticiones** de objetos

El método **add()** devuelve una ***referencia*** al **objeto distribuido recién creado**, al cual se le denomina ***proxy***

Se **registra** el ***sirviente*** en el ***adaptador*** mediante el método **add()** indicando la **identidad** del objeto (***printer1***)

Finalmente, en la función **main()** se crea una instancia de **Server** e invoca a su método **main()**

Indica a la aplicación que **termine** el **comunicador** al recibir la señal de *SIGQUIT* (control+C)

Bloquea el hilo principal hasta que el comunicador sea terminado

La **identidad** del objeto remoto **se ocupa en el cliente**

6

Client : El Cliente sólo necesita conseguir una ***referencia*** al objeto remoto para **invocar su método write()**

```
public class Client extends Ice.Application {
  public int run(String[] args) {

    Ice.ObjectPrx proxyObj = communicator().stringToProxy(args[0]);

    Example.PrinterPrx printer = Example.PrinterPrxHelper.checkedCast(proxyObj);

    printer.write("Hello, World!");

    return 0;
  }

  static public void main(String[] args) {
    Client app = new Client();
    app.main("Client", args);
  }
}
```

de esta línea se obtiene un objeto **proxy genérico (proxyObj)** el cual debe ser **casteado** (**downcasting**) a la clase de la interfaz Printer para así poder utilizar sus metodos

El programa acepta por **línea de comandos** la **representación textual** del **proxy** del **objeto remoto**

Para hacer el downcasting se utiliza el método **PrinterPrxHelper.checkedCast()**

Una vez conseguido el proxy del tipo correcto (objeto **printer**), se puede **invocar el método write()** pasando por parámetros, la cadena "bello world"