ICE

Internet Communications Engine

UNIVERSIDAD ICESI

What is Ice?

Is an object-oriented RPC framework/middleware that helps you build distributed applications with minimal effort.

With Ice, there is no need to worry about details such as opening network connections, serializing and deserializing data for network transmission or retrying failed connection attempts.

Language and Operating System Interoperability

Write your client in one language, your server in another, and run them on your favorite platforms. It's all seamless; your client and server are unaware of the language and platform the other is using.

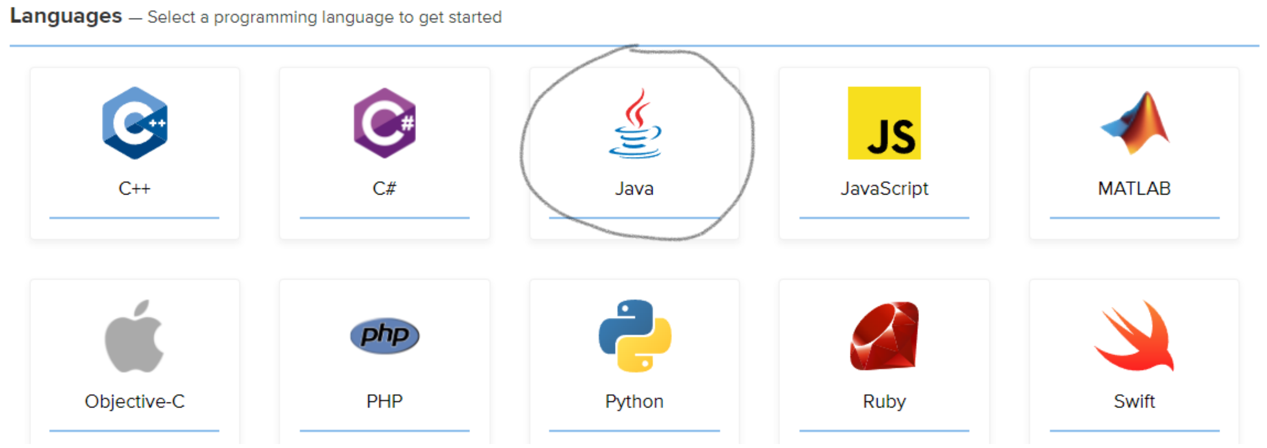
Installations.

Prerequisite

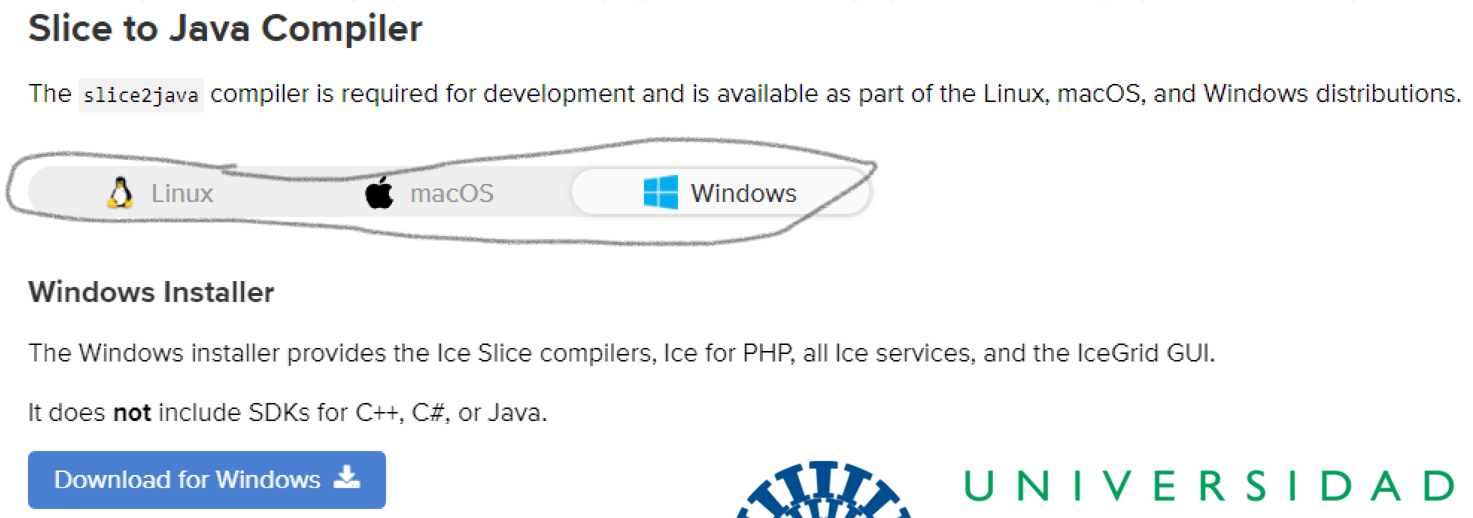
- Java jdk: version mayor a la 8. (11.0.8 LTS)
- Gradle: version 6.6

Down load ice

Link: https://zeroc.com/downloads/ice



Languages — Select a programming language to get started
C++
C#
Java
JavaScript
MATLAB
Objective-C
PHP
Python
Ruby
Swift



Slice to Java Compiler

The `slice2java` compiler is required for development and is available as part of the Linux, macOS, and Windows distributions.

Linux macOS Windows

Windows Installer

The Windows installer provides the Ice Slice compilers, Ice for PHP, all Ice services, and the IceGrid GUI.

It does **not** include SDKs for C++, C#, or Java.

Architecture.

Terminology.

- **Clients and Servers.**

  - *Clients are active entities. They issue requests for service to servers.*
  - *Servers are passive entities. They provide services in response to client requests.*

- **Ice Objects.**
  An Ice object is a conceptual entity, or abstraction. An Ice object can be characterized by the following points:

  - *An Ice object is an entity in the local or a remote address space that can respond to client requests.*
  - *Each Ice object has one or more interfaces. An interface is a collection of named operations that are supported by an object.*
  - *An operation has zero or more parameters as well as a return value. Parameters and return values have a specific type*
  - *Each Ice object has a unique object identity.*
  - *An Ice object has a distinguished interface, known as its main interface*

UNIVERSIDAD ICESI

Architecture.

Terminology.

- **Proxies.**

  A proxy is an artifact that is local to the client's address space; it represents the (possibly remote) Ice object for the client.

  Functions: *Locates the Ice object, Activates the Ice object's server if it is not running, Activates the Ice object within the server, Transmits any in-parameters to the Ice object, Waits for the operation to complete, Returns any out-parameters and the return value to the client (or throws an exception in case of an error)*

- **Properties**.

  Properties are name-value pairs, such as Ice.Default.Protocol=tcp. Properties are typically stored in text files and parsed by the Ice run time to configure various options.

Specification Language for Ice (Slice)

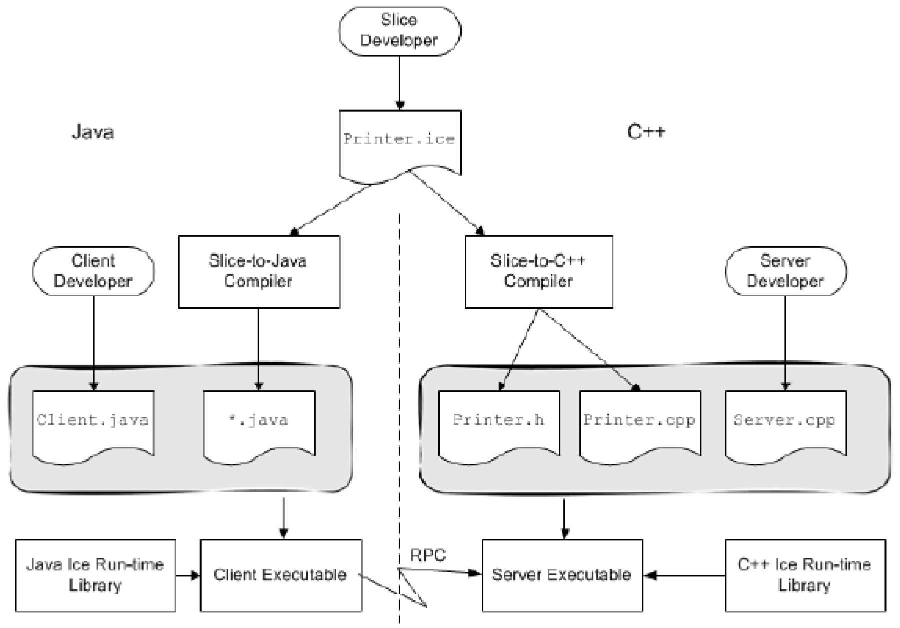
Each Ice object has an interface with a number of operations. Interfaces, operations, and the types of data that are exchanged between client and server are defined using the *Slice language*.

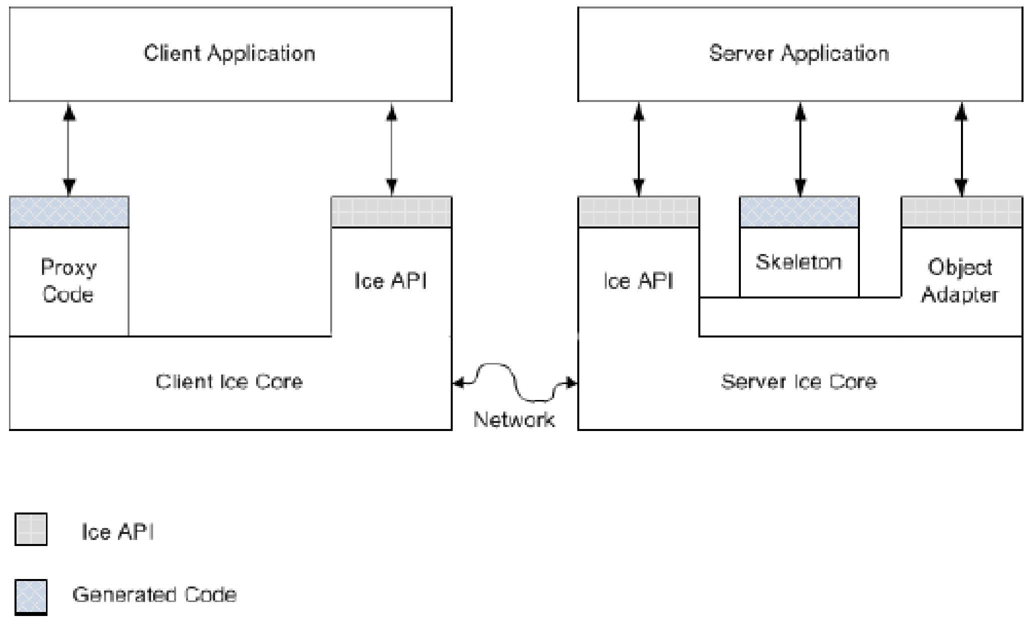
Slice allows you to define the client-server contract in a way that is independent of a specific programming language, such as C++, Java, or C#.

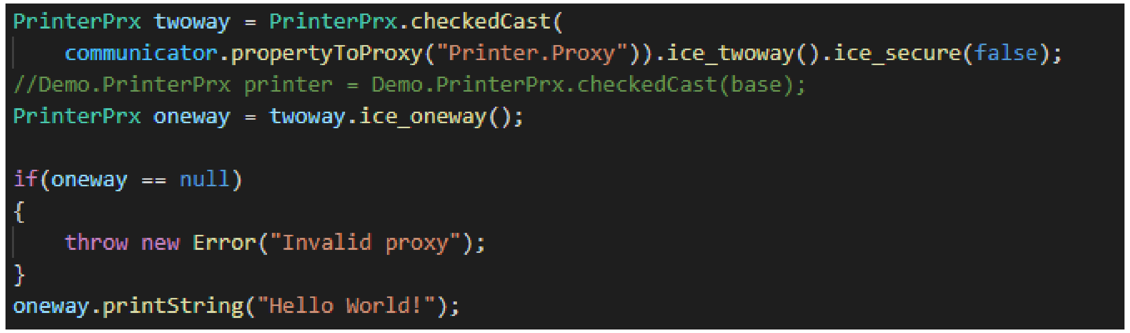
Slice describes interfaces and types (but not implementations), it is a purely declarative language; there is no way to write executable statements in Slice.

UNIVERSIDAD ICESI

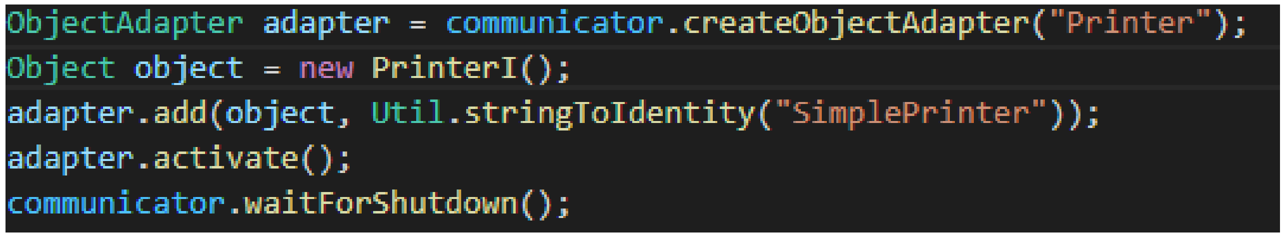
Specification Language for Ice (Slice)





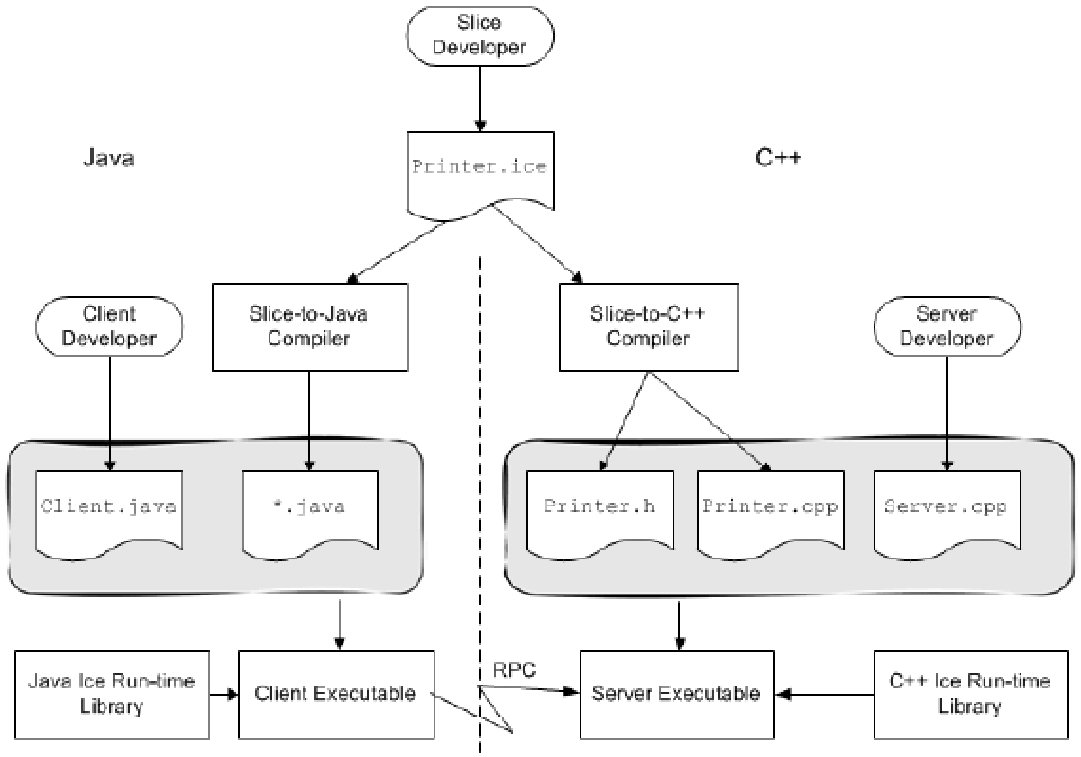


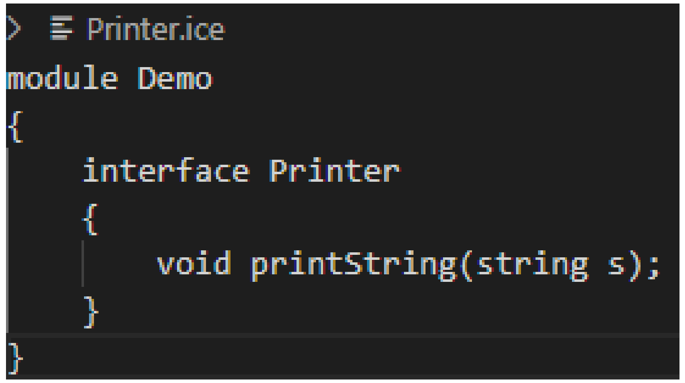
```
PrinterPrx twoway = PrinterPrx.checkedCast(
    communicator.propertyToProxy("Printer.Proxy")).ice_twoway().ice_secure(false);
//Demo.PrinterPrx printer = Demo.PrinterPrx.checkedCast(base);
PrinterPrx oneway = twoway.ice_oneway();

if(oneway == null)
{
    throw new Error("Invalid proxy");
}
oneway.printString("Hello World!");
```



```
ObjectAdapter adapter = communicator.createObjectAdapter("Printer");
Object object = new PrinterI();
adapter.add(object, Util.stringToIdentity("SimplePrinter"));
adapter.activate();
communicator.waitForShutdown();
```

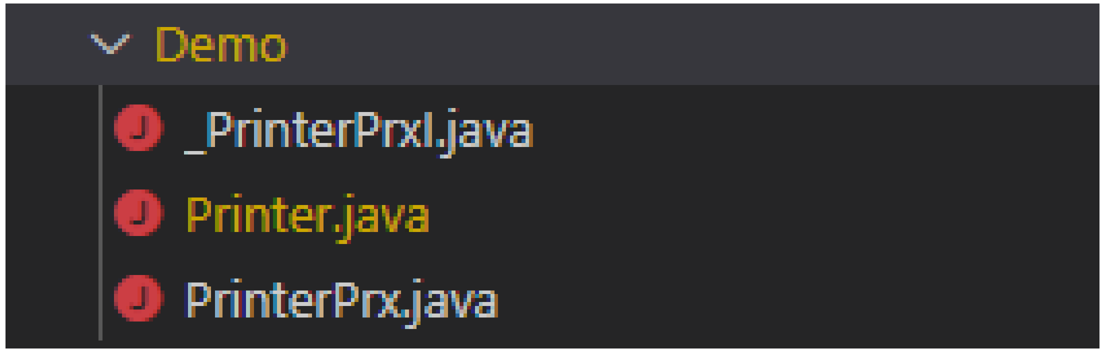
Specification Language for Ice (Slice)





Printer.ice

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



Demo
- _PrinterPrxI.java
- Printer.java
- PrinterPrx.java

Specification Language for Ice (Slice)

Lexical rules.

- Comments:

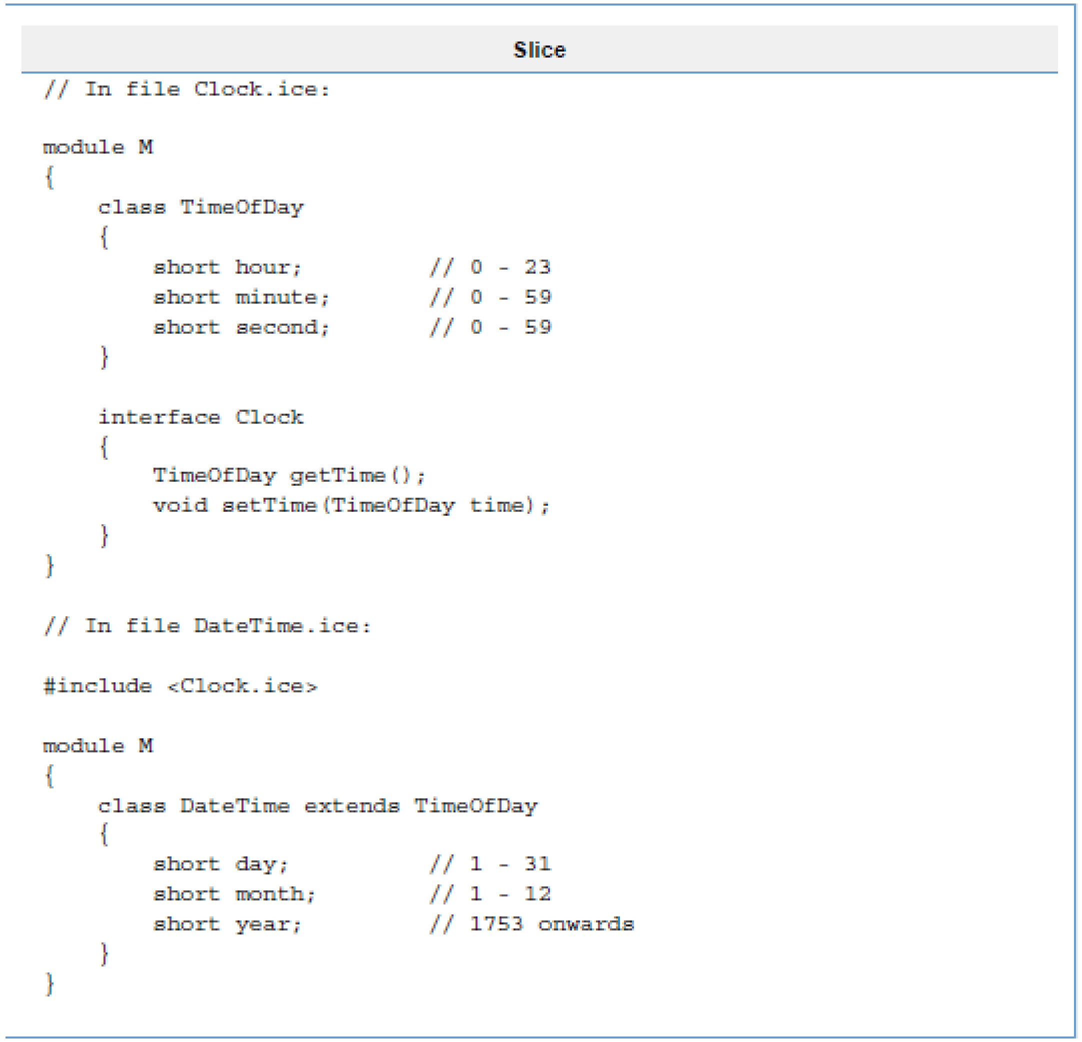
  /*
  * comment
  */ o
  // comment

- Modules:

  module modulName{
  modules, classes, interfaces, structures, types} java mapping like package moduleName

- Interface:
  interface interfaceName{
  Methods …}

- Classes:
  class className{
  attributes, methods}


Slice

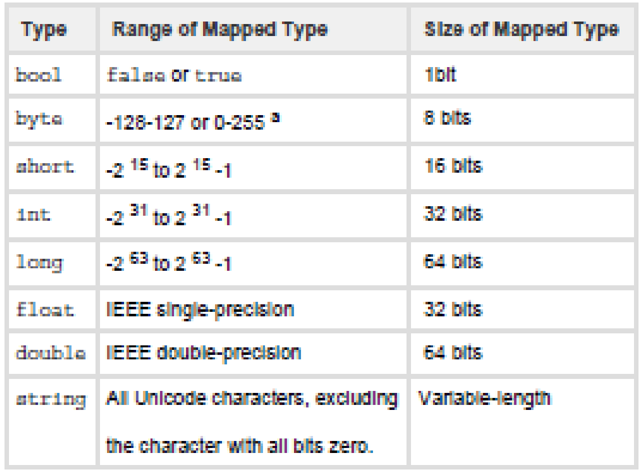
```
// In file Clock.ice:

module M
{
    class TimeOfDay
    {
        short hour;         // 0 - 23
        short minute;       // 0 - 59
        short second;       // 0 - 59
    }

    interface Clock
    {
        TimeOfDay getTime();
        void setTime(TimeOfDay time);
    }
}

// In file DateTime.ice:

#include <Clock.ice>

module M
{
    class DateTime extends TimeOfDay
    {
        short day;          // 1 - 31
        short month;        // 1 - 12
        short year;         // 1753 onwards
    }
}
```

Slice requires all definitions to be nested inside a module, that is, you cannot define anything other than a module at global scope.

Specification Language for Ice (Slice)

Lexical rules.

- Basic types:



Type	Range of Mapped Type	Size of Mapped Type
bool	false or true	1bit
byte	-128-127 or 0-255 [a]	8 bits
short	-2^{15} to $2^{15}-1$	16 bits
int	-2^{31} to $2^{31}-1$	32 bits
long	-2^{63} to $2^{63}-1$	64 bits
float	IEEE single-precision	32 bits
double	IEEE double-precision	64 bits
string	All Unicode characters, excluding the character with all bits zero.	Variable-length

- User defined types:
  - *Enums:* enum Fruit{ Apple=1, Pear=2, Orange=3}
  - *Structures:* struct Time{ short hour; }
  - *Sequences:* sequence<Fruit> fruits; → Fruit[] fruits;
  - *Dictionaries:* dictionary<key, value> map; → HashMap<key, value> map;
  - *Objects:* ["java:serializable:SomePackage.JavaClass"] sequence<byte> JavaObj;

Specification Language for Ice (Slice)

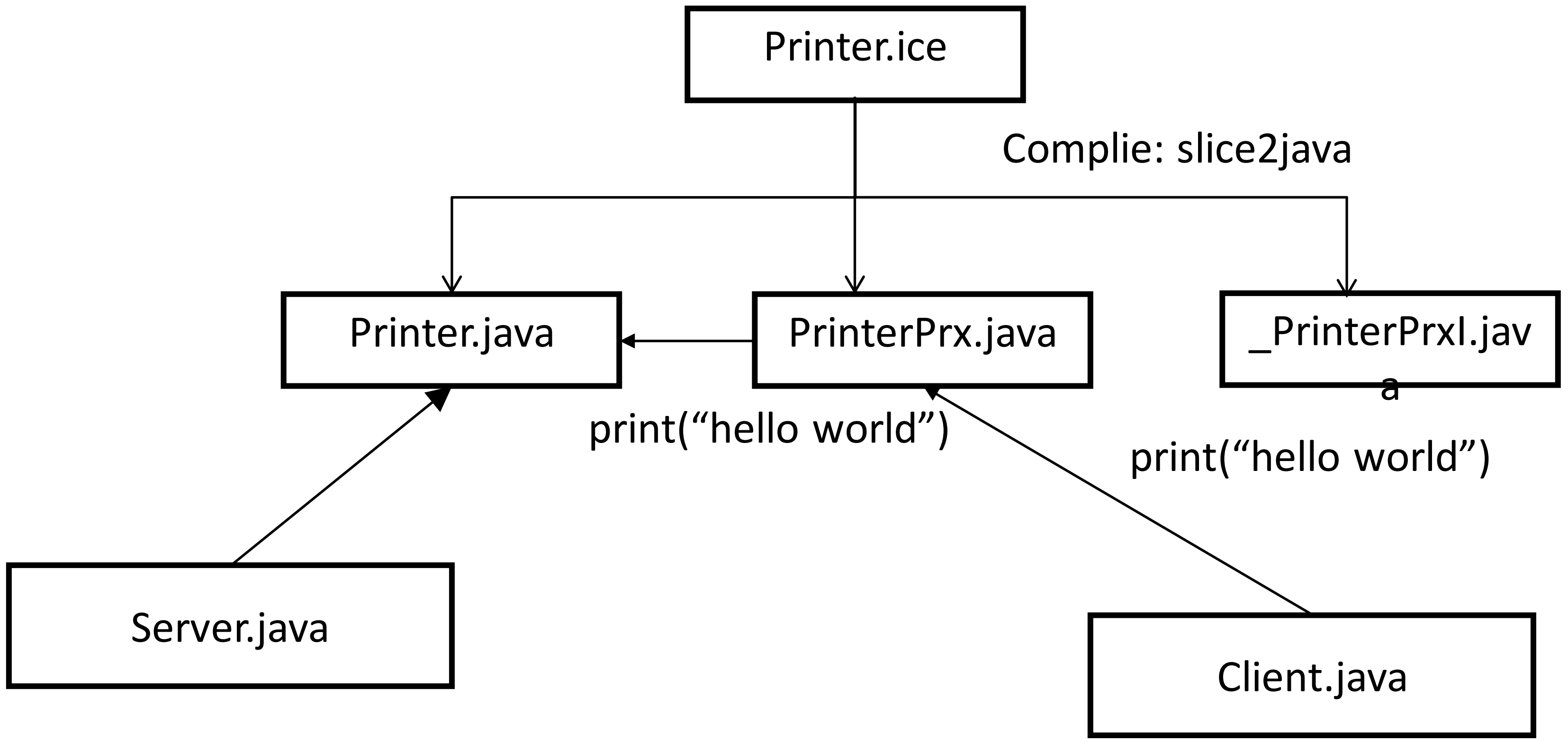
Lexical rules.

- Proxies for ice objects.

  *The * operator is*
  known as the proxy operator.

```
module Demo
{
   interface CallbackReceiver
   {
      void callback(string msg);
   }

   interface CallbackSender
   {
      void initiateCallback(CallbackReceiver* proxy, string msg);
    }
}
```

Hello World


Printer.ice
Complie: slice2java
Printer.java
PrinterPrx.java
_PrinterPrxI.java
print("hello world")
print("hello world")
Server.java
Client.java