

A gentle introduction to



Saverio Giallorenzo | sgiallor@cs.unibo.it

$$\frac{j \in I \quad t_c = \text{eval}(e, t) \quad M(t_c) = (o_j, t') :: \tilde{m}}{\sum_{i \in I} [o_i(x_i) \text{ from } e] \{B_i\} \cdot t \cdot M \rightarrow B_j \cdot t \triangleleft (x_j, t') \cdot M[t_c \mapsto \tilde{m}]} \quad \lfloor^{\text{DCC}|\text{Choice}} \rfloor$$

$$\frac{t' = \text{eval}(x, t)}{x = e; B \cdot t \cdot M \rightarrow B \cdot t \triangleleft (x, t') \cdot M} \quad \lfloor^{\text{DCC}|\text{Assign}} \rfloor$$

$$\frac{P \rightarrow P'}{P \mid P_1 \rightarrow P' \mid P_1} \quad \lfloor^{\text{DCC}|\text{PPar}} \rfloor$$

$$\frac{P = \text{cq}(x); B \cdot t \cdot M \quad t_c \notin \bigcup_i \text{dom}(M_i) \cup \text{dom}(M) \quad t' = t \triangleleft (x, t_c)}{\langle B_s, P \mid \prod_i B_i \cdot t_i \cdot M_i \rangle_l \rightarrow \langle B_s, B \cdot t' \cdot M[t_c \mapsto \varepsilon] \mid \prod_i B_i \cdot t_i \cdot M_i \rangle_l} \quad \lfloor^{\text{DCC}|\text{Cq}} \rfloor$$

$$\frac{P \equiv P_1 \quad P_1 \rightarrow P'_1 \quad P'_1 \equiv P'}{\langle B_s, P \rangle_l \rightarrow \langle B_s, P' \rangle_l} \quad \lfloor^{\text{DCC}|\text{PEq}} \rfloor$$

$$\frac{P = o @ e_1(e_2) \text{ to } e_3; B \cdot t \cdot M \quad \begin{array}{c} \text{eval}(e_1, t) = l \\ \text{eval}(e_2, t) = t_m \end{array} \quad \begin{array}{c} \text{eval}(e_3, t) = t_c \\ M'' = M'[t_c \mapsto M'(t_c) :: (o, t_m)] \end{array}}{\langle B_s, P \mid B' \cdot t' \cdot M' \mid P_1 \rangle_l \rightarrow \langle B_s, B \cdot t \cdot M \mid B' \cdot t' \cdot M'' \mid P_1 \rangle_l} \quad \lfloor^{\text{DCC}|\text{InSend}} \rfloor$$

$$\frac{P = o @ e_1(e_2) \text{ to } e_3; B \cdot t \cdot M \quad \begin{array}{c} \text{eval}(e_1, t) = l' \\ \text{eval}(e_2, t) = t_m \end{array} \quad \begin{array}{c} \text{eval}(e_3, t) = t_c \\ M'' = M'[t_c \mapsto M'(t_c) :: (o, t_m)] \end{array}}{\langle B_s, P \mid P_1 \rangle_l \mid \langle B'_s, B' \cdot t' \cdot M' \mid P_2 \rangle_{l'} \rightarrow \langle B_s, B \cdot t \cdot M \mid P_1 \rangle_l \mid \langle B'_s, B' \cdot t' \cdot M'' \mid P_2 \rangle_{l'}} \quad \lfloor^{\text{DCC}|\text{Send}} \rfloor$$

$$\frac{P_1 = ? @ e_1(e_2); B_1 \cdot t_1 \cdot M_1 \quad \text{eval}(e_1, t_1) = l \quad Q = B \cdot t_\perp \triangleleft (x, \text{eval}(e_2, t_1)) \cdot \emptyset}{\langle !(x); B, P \rangle_l \mid \langle B'_s, P_1 \mid P_2 \rangle_{l'} \rightarrow \langle !(x); B, Q \mid P \rangle_l \mid \langle B'_s, B_1 \cdot t_1 \cdot M_1 \mid P_2 \rangle_{l'}} \quad \lfloor^{\text{DCC}|\text{Start}} \rfloor$$

FORMAL CALCULUS
(like POCS and
the pi-calculus)

$$\frac{j \in I \quad t_c = \text{eval}(e, t) \quad M(t_c) = (o_j, t') :: \tilde{m}}{\sum_{i \in I} [o_i(x_i) \text{ from } e] \{B_i\} \cdot t \cdot M \rightarrow B_j \cdot t \triangleleft (x_j, t') \cdot M[t_c \mapsto \tilde{m}]} \quad [\text{DCC}|\text{Choice}]$$

$$\frac{t' = \text{eval}(x, t)}{x = e; B \cdot t \cdot M \rightarrow B \cdot t \triangleleft (x, t')} \quad \frac{P \rightarrow P'}{P \mid P'} \quad [\text{DCC}|\text{Cq}]$$

$$\frac{P = \text{cq}(x); B \cdot t \cdot M}{\langle B_s, P \mid \prod B \rangle_l} \quad [\text{DCC}|\text{Cq}]$$

$$\frac{P = e}{\langle B_s, P \mid P \rangle_l} \quad [\text{DCC}|\text{Send}]$$

$$\frac{P = o; \text{eval}(e_2, t_2) :: \tilde{m}; \dots; t_c \mapsto l'; \text{eval}(e_3, t) = t_c; M'(t_c) :: (o, t_m)]}{\langle B_s, P \mid P_1 \rangle_l \mid \langle B'_s, B \cdot t \cdot M \mid P_1 \rangle_l \mid \langle B'_s, B' \cdot t' \cdot M'' \mid P_2 \rangle_{l'}} \quad [\text{DCC}|\text{Send}]$$

$$\frac{P_1 = ?@e_1(e_2); B_1 \cdot t_1 \cdot M_1 \mid P_1 \quad \text{eval}(e_1, t_1) = l \quad Q = B \cdot t_\perp \triangleleft (x, \text{eval}(e_2, t_1)) \cdot \emptyset}{\langle !(x); B, P \rangle_l \mid \langle B'_s, P_1 \mid P_2 \rangle_{l'} \rightarrow \langle !(x); B, Q \mid P \rangle_l \mid \langle B'_s, B_1 \cdot t_1 \cdot M_1 \mid P_2 \rangle_{l'}} \quad [\text{DCC}|\text{Start}]$$

What is Jolie?

A Service-Oriented Programming Language

Service-Oriented

Service
Instances

Operations

Object-Oriented

Objects

Methods

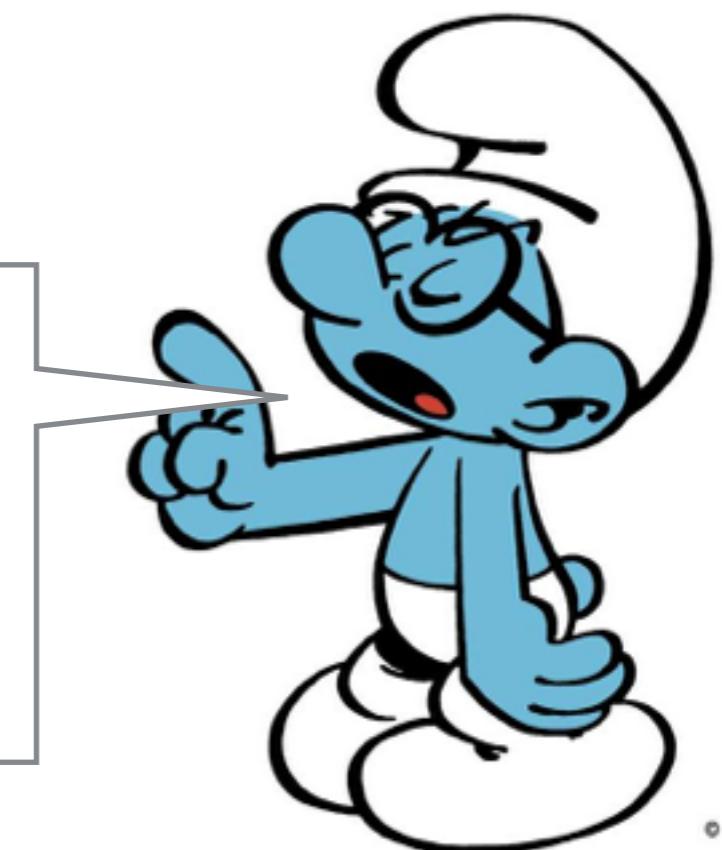
Why SOC and Jolie?

Jolie is perfect for fast prototyping. In little time a small team of developers can build up a full-fledged distributed system.

Why SOC and Jolie?

Jolie is perfect for fast prototyping. In little time a small team of developers can build up a full-fledged distributed system.

But I already know Java!
Why shall I use Jolie?



Why SOC and Jolie?

Why SOC and Jolie?



Why SOC and Jolie?

```
SocketChannel socketChannel = SocketChannel.open();
socketChannel.connect(
new InetSocketAddress("http://someurl.com", 80));
Buffer buffer = . . .; // byte buffer
while( buffer.hasRemaining() ) {
    channel.write( buffer );
}
```

Happy?



Why SOC and Jolie?

```
SocketChannel socketChannel = SocketChannel.open();
socketChannel.connect(
new InetSocketAddress("http://someurl.com", 80));
Buffer buffer = . . .; // byte buffer
while( buffer.hasRemaining() ) {
    channel.write( buffer );
}
```

Happy?

Ok, but you did not even close
the channel or handled
exceptions



Why SOC and Jolie?



Why SOC and Jolie?

```
SocketChannel socketChannel = SocketChannel.open();
try {
    socketChannel.connect(new InetSocketAddress("http://someurl.com",
80));
    Buffer buffer = . . .; // byte buffer
    while( buffer.hasRemaining() ) {
        channel.write( buffer );
    }
} catch( UnresolvedAddressException e ) { . . . }
catch( SecurityException e ) { . . . }
/* . . . many catches later . . . */
catch( IOException e ) { . . . }
finally { channel.close(); }
```

Happier now?



Why SOC and Jolie?

```
SocketChannel socketChannel = SocketChannel.open();
try {
    socketChannel.connect(new InetSocketAddress("http://someurl.com",
80));
    Buffer buffer = . . .; // byte buffer
    while( buffer.hasRemaining() ) {
        channel.write( buffer );
    }
} catch( UnresolvedAddressException e ) { . . . }
catch( SecurityException e ) { . . . }
/* . . . many catches later . . . */
catch( IOException e ) { . . . }
finally { channel.close(); }
```

Happier now?



Yes, but what about the
server?

Why SOC and Jolie?

Why SOC and Jolie?



Why SOC and Jolie?

```
Selector selector = Selector.open();
channel.configureBlocking(false);
SelectionKey key = channel.register(selector, SelectionKey.OP_READ);
while(true) {
    int readyChannels = selector.select();
    if(readyChannels == 0) continue;
    Set<SelectionKey> selectedKeys = selector.selectedKeys();
    Iterator<SelectionKey> keyIterator = selectedKeys.iterator();
    while(keyIterator.hasNext()) {
        SelectionKey key = keyIterator.next();
        if(key.isAcceptable()) {
            // a connection was accepted by a ServerSocketChannel.
        } else if (key.isConnectable()) {
            // a connection was established with a remote server.
        } else if (key.isReadable()) {
            // a channel is ready for reading
        } else if (key.isWritable()) {
            // a channel is ready for writing
        }
        keyIterator.remove();
    }
}
```

Here you are



Why SOC and Jolie?

Well, ok, but again, you are not **handling exceptions**.
And what about if **different operations** use the **same channel**?

And if we wanted to use **RMIs** instead of **Sockets**?

In what **format** are you transmitting data? And if we need to **change** the **format** after we wrote the application? Do you **check** the **type of data** you receive/send?

Why SOC and Jolie?

Well, ok, but again, you are not **handling exceptions**.
And what about if **different operations** use the **same channel**?

And if we wanted to use **RMIs** instead of **Sockets**?

In what **format** are you transmitting data? And if we need to **change** the **format** after we wrote the application? Do you **check** the **type of data** you receive/send?



Why SOC and Jolie?

Programming distributed systems is usually harder than programming non distributed ones.

Concerns of **concurrent** programming.

Plus (not exhaustive):

- handling **communications**;
- handling **heterogeneity**;
- handling **faults**;
- handling the **evolution** of systems.

Hello World! in Jolie

Let us get our hands dirty.

“Hello World!” is enough to let you see some of the main features of Jolie and Service-Oriented Programming.

```
include "console.iol"

main
{
    println@Console( "Hello, world!" )()
}
```

Hello World! in Jolie

Let us get our hands dirty.

“Hello World!” is enough to let you see some of the main features of Jolie and Service-Oriented Programming.

```
include "console.iol"

main
{
    println@Console( "Hello, world!" )()
}
```

Hello World! in Jolie

Let us get our hands dirty.

“Hello World!” is enough to let you see some of the main features of Jolie and Service-Oriented Programming.

```
include "console.iol"  
  
main  
{  
    println@Console( "Hello, world!" )()  
}
```



Hello World! in Jolie

Let us get our hands dirty.

“Hello World!” is enough to let you see some of the main features of Jolie and Service-Oriented Programming.

```
include "console.iol"
main
{
    println@Console( "Hello, world!" )()
}
```



Include a service

Hello World! in Jolie

Let us get our hands dirty.

“Hello World!” is enough to let you see some of the main features of Jolie and Service-Oriented Programming.

```
include "console.iol"
main
{
    println@Console( "Hello, world!" )()
}
```



Include a service

Hello World! in Jolie

Let us get our hands dirty.

“Hello World!” is enough to let you see some of the main features of Jolie and Service-Oriented Programming.

```
include "console.iol"
main
{
    println@Console( "Hello, world!" )()
}
```

Include a service

main

Hello World! in Jolie

Let us get our hands dirty.

“Hello World!” is enough to let you see some of the main features of Jolie and Service-Oriented Programming.

```
include "console.iol"
main
{
    println@Console( "Hello, world!" )()
}
```

The code snippet shows a Jolie script. It starts with an `include` statement for the `"console.iol"` service. The `main` block contains a single line of code: `println@Console("Hello, world!")()`. Annotations with arrows explain the code: a blue arrow points from the text "Include a service" to the `include` statement, and another blue arrow points from the text "program entry point" to the `main` keyword.

Hello World! in Jolie

Let us get our hands dirty.

“Hello World!” is enough to let you see some of the main features of Jolie and Service-Oriented Programming.

```
include "console.iol"
main
{
    println@Console( "Hello, world!" )()
}
```

The code snippet shows a Jolie script. It starts with an `include` statement for the `"console.iol"` service. The `main` block contains a single line of code: `println@Console("Hello, world!")()`. Annotations with arrows explain the code: a blue arrow points from the text "Include a service" to the `include` statement, and another blue arrow points from the text "program entry point" to the `main` keyword.

Hello World! in Jolie

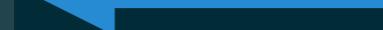
Let us get our hands dirty.

“Hello World!” is enough to let you see some of the main features of Jolie and Service-Oriented Programming.

```
include "console.iol"
main
{
    println@Console( "Hello, world!" )()
}
```

Include a service

program entry point



Hello World! in Jolie

Let us get our hands dirty.

“Hello World!” is enough to let you see some of the main features of Jolie and Service-Oriented Programming.

```
include "console.iol"
main
{
    println@Console( "Hello, world!" )()
}
```

The diagram illustrates the structure of the Jolie code. A blue arrow points from the text 'Include a service' to the line 'include "console.iol"'. Another blue arrow points from the text 'program entry point' to the word 'main'. A third blue arrow points from the text 'operation' to the annotated line 'println@Console("Hello, world!")()'.

Hello World! in Jolie

Let us get our hands dirty.

“Hello World!” is enough to let you see some of the main features of Jolie and Service-Oriented Programming.

```
include "console.iol"
main
{
    println@Console( "Hello, world!" )()
}
```

The diagram illustrates the structure of the Jolie code. A blue arrow points from the text 'Include a service' to the line 'include "console.iol"'. Another blue arrow points from the text 'program entry point' to the word 'main'. A third blue arrow points from the text 'operation' to the line 'println@Console("Hello, world!")()'.

Hello World! in Jolie

Let us get our hands dirty.

“Hello World!” is enough to let you see some of the main features of Jolie and Service-Oriented Programming.

```
include "console.iol"
main
{
    println@Console( "Hello, world!" )()
}
```

Include a service

program entry point

operation

The diagram illustrates a Jolie program with the following code:

```
include "console.iol"
main
{
    println@Console( "Hello, world!" )()
}
```

Annotations with blue arrows explain the components:

- An arrow labeled "Include a service" points to the `include "console.iol"` statement.
- An arrow labeled "program entry point" points to the `main` keyword.
- Two arrows labeled "operation" point to the `println` statement.

Hello World! in Jolie

Let us get our hands dirty.

“Hello World!” is enough to let you see some of the main features of Jolie and Service-Oriented Programming.

```
include "console.iol"
main
{
    println@Console( "Hello, world!" )()
}
```

The diagram illustrates the structure of the Jolie code. A blue arrow points from the text 'Include a service' to the 'include' statement. Another blue arrow points from the text 'program entry point' to the 'main' function. Two blue arrows point upwards from the bottom of the slide to the 'operation' and 'service' labels, which are positioned below the code block. The code itself is displayed in a dark teal background with white text, showing the inclusion of a service, the main entry point, the operation (println), and the service (Console) being used.

Include a service

program entry point

operation

service

Hello World! in Jolie

Let us get our hands dirty.

“Hello World!” is enough to let you see some of the main features of Jolie and Service-Oriented Programming.

```
include "console.iol"

main
{
    println@Console( "Hello, world!" )()
}
```

Hello World! in Jolie

Let us get our hands dirty.

“Hello World!” is enough to let you see some of the main features of Jolie and Service-Oriented Programming.



Hello World! in Jolie

Let us get our hands dirty.

“Hello World!” is enough to let you see some of the main features of Jolie and Service-Oriented Programming.



hello_world.ol

Hello World! in Jolie

Let us get our hands dirty.

“Hello World!” is enough to let you see some of the main features of Jolie and Service-Oriented Programming.

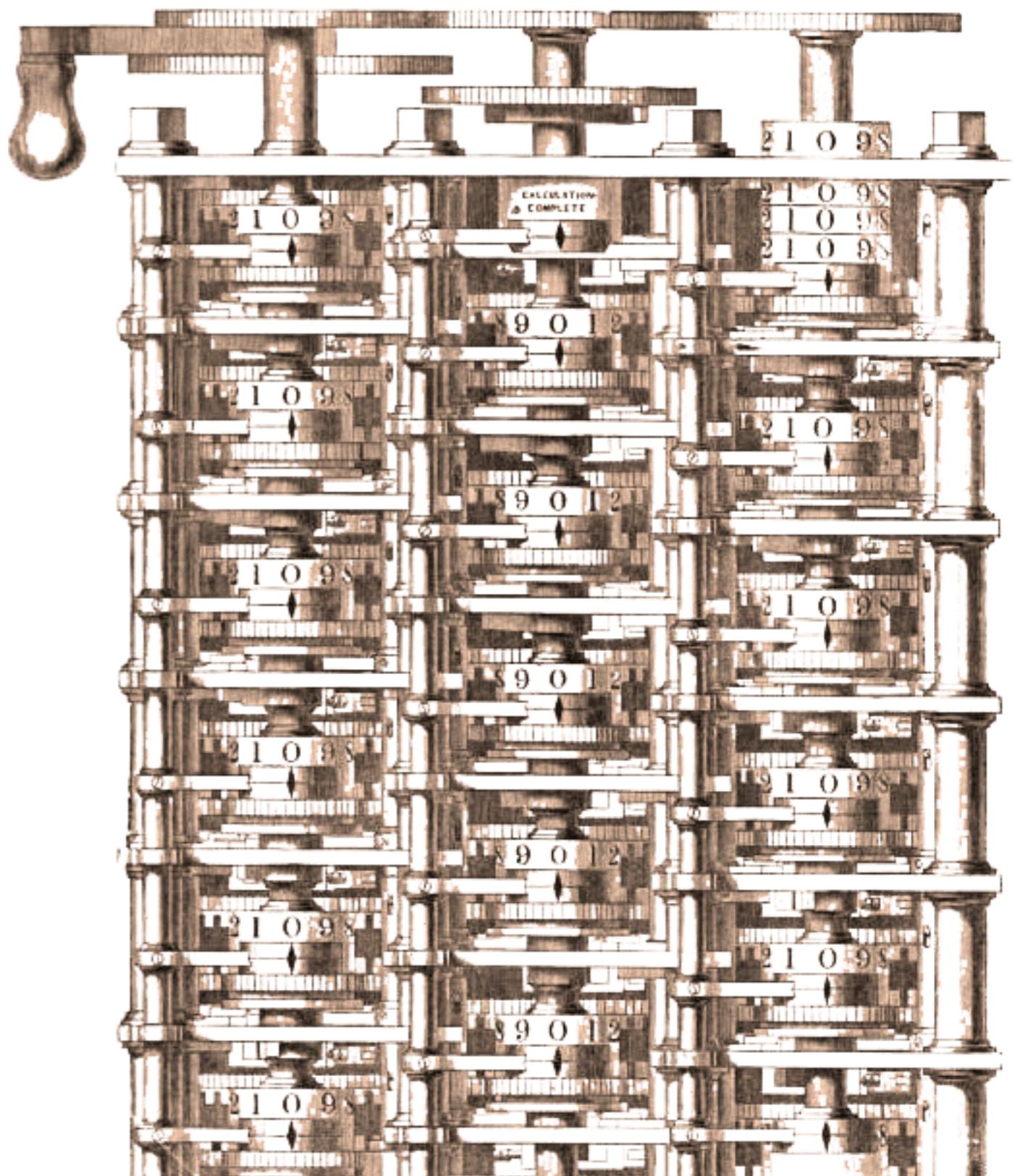


```
$ jolie hello_world.ol
```

hello_world.ol

Let us see some Jolie in Action

Everything starts
with a **calculator**...



Behaviours and Deployments

```
interface MyInterface {  
    OneWay: sendNumber( int )  
}
```

```
include "MyInterface.iol"  
outputPort B {  
    Location:  
        "socket://localhost:8000"  
    Protocol: sodep  
    Interfaces: MyInterface  
}  
  
main  
{  
    sendNumber @ B ( 5 )  
}
```

Client

```
include "MyInterface.iol"  
inputPort B {  
    Location:  
        "socket://localhost:8000"  
    Protocol: sodep  
    Interfaces: MyInterface  
}  
  
main  
{  
    sendNumber( x )  
}
```

Server

Behaviours and Deployments

```
interface MyInterface {  
    OneWay: sendNumber( int )  
}
```

```
include "MyInterface.iol"  
outputPort B {  
    Location:  
        "socket://localhost:8000"  
    Protocol: sodep  
    Interfaces: MyInterface  
}  
  
main  
{  
    sendNumber @ B ( 5 )  
}
```

Client

```
include "MyInterface.iol"  
inputPort B {  
    Location:  
        "socket://localhost:8000"  
    Protocol: sodep  
    Interfaces: MyInterface  
}  
  
main  
{  
    sendNumber( x )  
}
```

Server

Behaviours and Deployments

```
interface MyInterface {  
    OneWay: sendNumber( int )  
}
```

```
include "MyInterface.iol"  
outputPort B {  
    Location:  
        "socket://localhost:8000"  
    Protocol: sodep  
    Interfaces: MyInterface  
}  
  
main  
{  
    sendNumber @ B ( 5 )  
}
```

Client

```
include "MyInterface.iol"  
inputPort B {  
    Location:  
        "socket://localhost:8000"  
    Protocol: sodep  
    Interfaces: MyInterface  
}  
  
main  
{  
    sendNumber( x )  
}
```

Server

Behaviours and Deployments

```
interface MyInterface {  
    OneWay: sendNumber( int )  
}
```

```
include "MyInterface.iol"  
outputPort B {  
    Location:  
        "socket://localhost:8000"  
    Protocol: sodep  
    Interfaces: MyInterface  
}  
  
main  
{  
    sendNumber @ B ( 5 )  
}
```

Client

```
include "MyInterface.iol"  
inputPort B {  
    Location:  
        "socket://localhost:8000"  
    Protocol: sodep  
    Interfaces: MyInterface  
}  
  
main  
{  
    sendNumber( x )  
}
```

Server

Behaviours and Deployments

```
interface MyInterface {  
    OneWay: sendNumber( int )  
}
```

```
include "MyInterface.iol"  
outputPort B {  
    Location:  
        "socket://localhost:8000"  
    Protocol: sodep  
    Interfaces: MyInterface  
}  
  
main  
{  
    sendNumber @ B( 5 )  
}
```

Client

```
include "MyInterface.iol"  
inputPort B {  
    Location:  
        "socket://localhost:8000"  
    Protocol: sodep  
    Interfaces: MyInterface  
}  
  
main  
{  
    sendNumber( x )  
}
```

Server

Behaviours and Deployments

```
interface MyInterface {  
    OneWay: sendNumber( int )  
}
```

```
include "MyInterface.iol"  
outputPort B {  
    Location:  
        "socket://localhost:8000"  
    Protocol: sodep  
    Interfaces: MyInterface  
}  
  
main  
{  
    sendNumber @ B ( 5 )  
}
```

Client

```
include "MyInterface.iol"  
inputPort B {  
    Location:  
        "socket://localhost:8000"  
    Protocol: sodep  
    Interfaces: MyInterface  
}  
  
main  
{  
    sendNumber( x )  
}
```

Server

Behaviours and Deployments

```
interface MyInterface {  
    OneWay: sendNumber( int )  
}
```

```
include "MyInterface.iol"  
outputPort B {  
    Location:  
        "socket://localhost:8000"  
    Protocol: sodep  
    Interfaces: MyInterface  
}  
  
main  
{  
    sendNumber @ B ( 5 )  
}
```

Client

```
include "MyInterface.iol"  
inputPort B {  
    Location:  
        "socket://localhost:8000"  
    Protocol: sodep  
    Interfaces: MyInterface  
}  
  
main  
{  
    sendNumber( x )  
}
```

Server

Behaviours and Deployments

```
interface MyInterface {  
    OneWay: sendNumber( int )  
}
```

```
include "MyInterface.iol"  
outputPort B {  
    Location:  
        "socket://localhost:8000"  
    Protocol: sodep  
    Interfaces: MyInterface  
}  
  
main  
{  
    sendNumber @ B ( 5 )  
}
```

Client

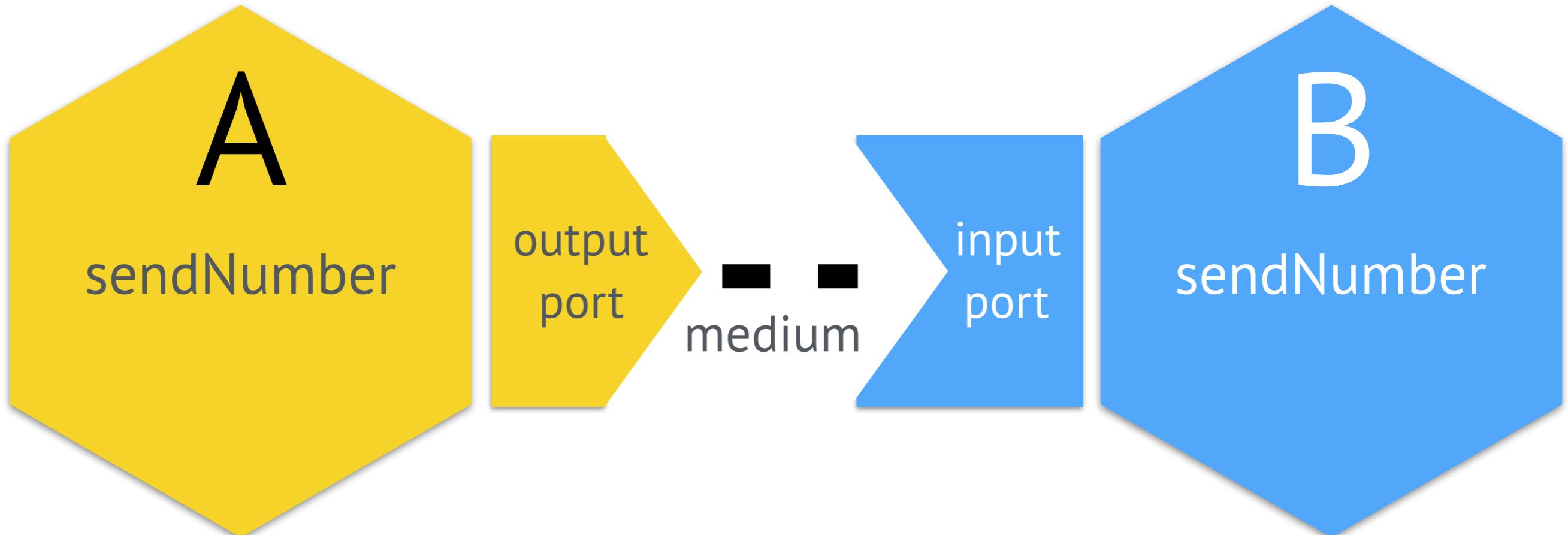
```
include "MyInterface.iol"  
inputPort B {  
    Location:  
        "socket://localhost:8000"  
    Protocol: sodep  
    Interfaces: MyInterface  
}  
  
main  
{  
    sendNumber( x )  
}
```

Server

Deployments

Enabling Communication

Behaviours and Deployments



- Services communicate through **ports**.
- **Ports** give access to an **interface**.
- An interface is a set of **operations**.
- An **output port** is used to invoke interfaces exposed by other services.
- An **input port** is used to expose an interface.

A closer look on ports - Locations

A location describes:

- the **communication medium**;
- the **parameters** to set the communication up.

In Jolie a **location** is a **Uniform Resource Identifier (URI)**
with form: **medium[:parameters]**



	Medium	Parameters
TCP/IP	<code>socket://</code>	<code>www.google.it:80</code>
Bluetooth	<code>btl2cap://</code>	<code>localhost:3B9FA89520078C303355AAA694238F07;name=Vision;encrypt=false;authenticate=false</code>
Local	<code>localsocket:</code>	<code>/tmp/mysocket.socket</code>
Java RMI	<code>rmi://</code>	<code>myRmiUrl.com/MyService</code>
In-Memory	<code>local</code>	

A closer look on ports - Protocols

A protocol defines the format the data is sent (**encoded**) and received (**encoded**)

In Jolie protocols are names and possibly additional parameters:

json/rpc

sodep

https

soap

http { .debug = true }

Behaviours

Composing Interactions

Interactions via Operations

Input
Operations

```
oneWay( req )  
  
reqRes( req )( res ){  
    // code block  
}
```

Output
Operations

```
oneWay@Port( req )  
  
reqRes@Port( req )( res )
```

Behaviour Composition

The sequence operator ; denotes that the **left operand** of the statement is executed **before** the one on the right.

```
println@Console( "A" )○;  
println@Console( "B" )○
```

Prints

A
B

Behaviour Composition

The parallel operator | states
that both left and right
operands execute concurrently

```
println@Console( "A" )○|  
println@Console( "B" )○
```

can print



but also



Behaviour Composition

The input choice implements **input-guarded non-deterministic choice**.

```
[ oneWayOperation() ] { branch_code }
[ oneWayOperation2() ] {branch_code2}

[ requestResponseOperation()
rr_code ]
] { branch_code }
```

Behaviour Composition

The input choice implements input-guarded non-deterministic choice.

```
main {
    [ buy( stock )( response ) {
        buy@Exchange( stock )( response )
    } ] { println@Console( "Buy order forwarded" )() }

    [ sell( stock )( response ) {
        sell@Exchange( stock )( response )
    } ] { println@Console( "Sell order forwarded" )() }
}
```

Last stand - that ORC example

```
include "net.inc"

val BingSpell =
    BingSpellFactoryPropertyFile
    ("orc/orchard/orchard.properties")
Println(y)
< y <
( Prompt("Input a string: ") > x >
  ( BingSpell(x) | (Rwait(250) >> x) ) )
```

Last stand - that ORC example

```
include "console.iol"
include "time.iol"

timeout = 250;
timeout.operation = "timeout";
txt = "Beutiful";
{
    spellCheck@BingSpell({ .text = txt, .location = myLoc })
    |
    setNextTimeout@Time( timeout )
};

[ spellCheckResponse( text )]{ println@Console( text )O }
[ timeout() ]{ throw( TimeoutException ) }
```

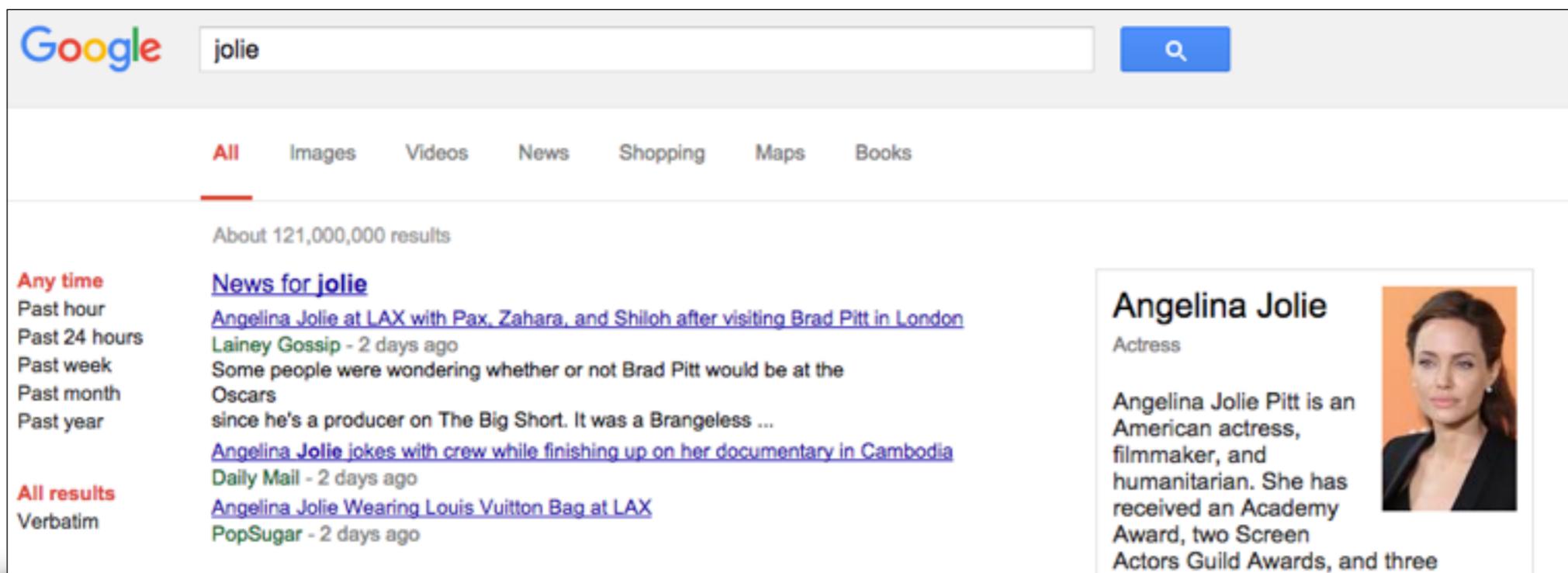


Before you take off

Jolie Website

<http://www.jolie-lang.org>

still working out the SEO...



A screenshot of a Google search results page for the query "jolie". The search bar at the top contains "jolie". Below it, a navigation bar includes "All" (which is highlighted in red), "Images", "Videos", "News", "Shopping", "Maps", and "Books". A sidebar on the left shows search filters: "Any time", "Past hour", "Past 24 hours", "Past week", "Past month", "Past year", and "All results". It also includes a "Verbatim" option. The main search results area displays "About 121,000,000 results". The first result is a link to "News for jolie" with a snippet about Angelina Jolie at LAX. The second result is from "Lainey Gossip" and mentions Brad Pitt. The third result discusses Brad Pitt's absence from the Oscars. The fourth result is from "Daily Mail" about Angelina Jolie's documentary in Cambodia. The fifth result is about Angelina Jolie wearing a Louis Vuitton bag. On the right side of the results, there is a large image of Angelina Jolie and a brief bio: "Angelina Jolie Pitt is an American actress, filmmaker, and humanitarian. She has received an Academy Award, two Screen Actors Guild Awards, and three".

The Jolie Interpreter

Last release

<http://www.jolie-lang.org/downloads.html>

- Requires JRE 1.6+
- Download jolie-installer.jar
- open a console and run

```
java -jar jolie-installer.jar
```

Sources

Jolie is an **open source** project with continuous updates and a well documented codebase

<https://github.com/jolie/jolie>

“This *is* the programming language you are looking for”



Documentation

Comprehensive and ever-growing
documentation and Standard Library.

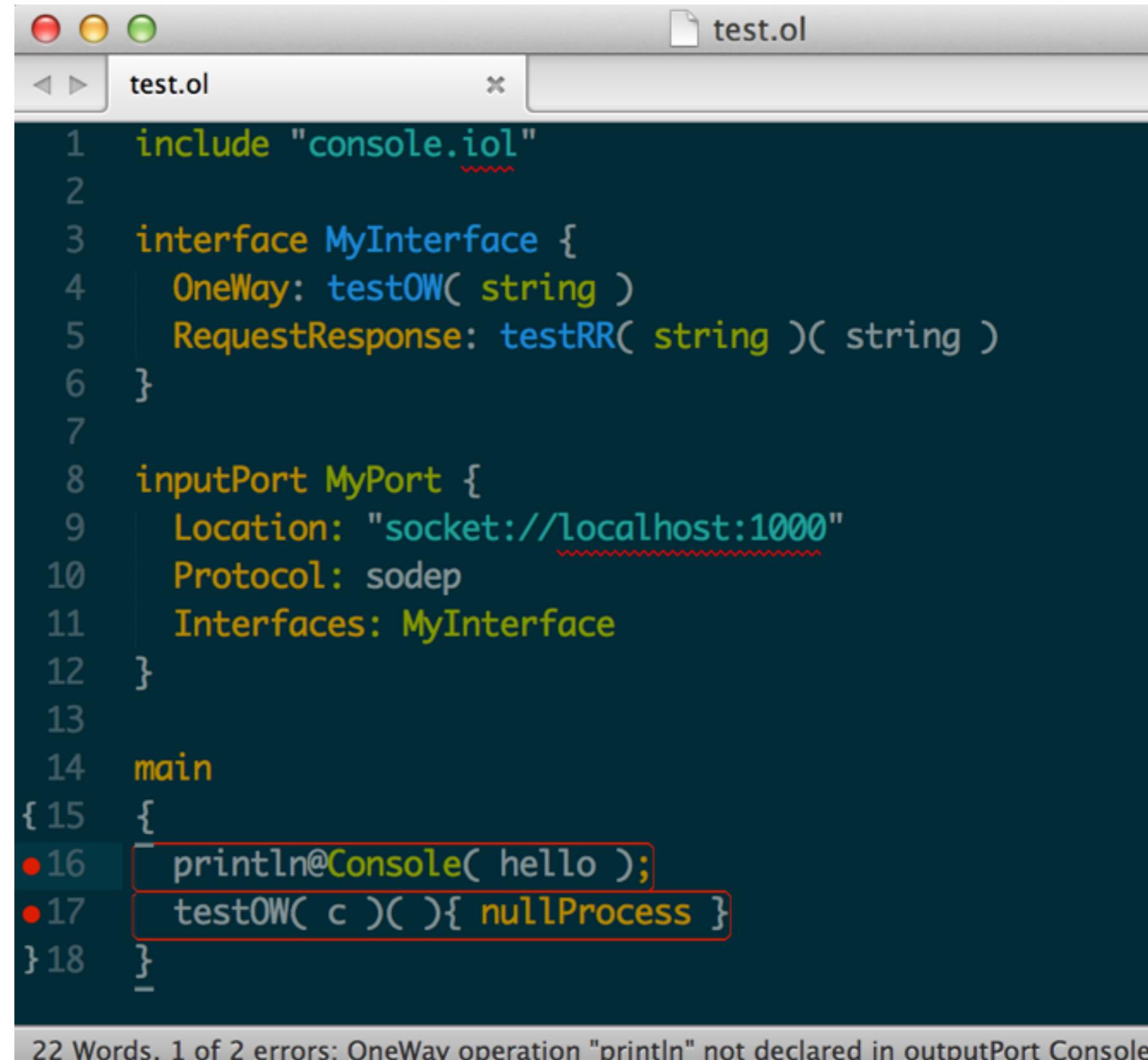
<http://docs.jolie-lang.org>



Editors

Sublime
Text but
also Atom

Syntax highlight,
online checking,
etc.



The screenshot shows a Sublime Text editor window titled "test.ol". The file contains Jolie code with the following content:

```
1 include "console.iol"
2
3 interface MyInterface {
4     OneWay: testOW( string )
5     RequestResponse: testRR( string )( string )
6 }
7
8 inputPort MyPort {
9     Location: "socket://localhost:1000"
10    Protocol: sodep
11    Interfaces: MyInterface
12 }
13
14 main
15 {
16     println@Console( hello );
17     testOW( c )( ){ nullProcess }
18 }
```

The code uses color-coded syntax highlighting. The "Location" field in the inputPort definition is underlined with a red wavy line, indicating an error. The "println" and "nullProcess" lines are highlighted with a red rectangle, likely indicating they are selected or part of a larger error range.

At the bottom of the editor, a status bar displays the message: "22 Words, 1 of 2 errors: OneWay operation "println" not declared in outputPort Console".



Thanks for your time!