What is it ?

A Driver is something taking a Ware and makeing somthing useful with it (typically generating source).

A Driver may be helped by NodeDrivers?, thus having a decentralized generation of products.

Services

Through class (ie even with no instance):

• Keeps a list of known NodeDrivers? and the associated <u>node</u> types. (see #Registering)

Through instances:

- Has some parameters (output directory, optional featues, ...) chosen at instanciation
- Keeps silent until code generation is asked (it may never be asked)

This way, we may generate with the same driver class (eg generate !SystemC/Caba code) different <u>Wares</u>, with different options; or the other way around, generate the same <u>Wares</u> many times through different backends.

Registering

With method register() a NodeDriver? can register as a generation class for a <u>Node</u>. API between a Driver and its associated NodeDrivers? is ad-hoc: the Driver's needs and design rules.

A base NodeDriver? valid for a given Driver should be given, with basic functionalities, doing a no-op for the code.

Registration usage

Let's have the following class hierarchies:

```
Node
MyWareTypeNode
MyWareWidgetNode
MyWareThingieNode
MyWareOtherNode
```

Let's have the following drivers:

```
NodeDriver
MyDriverNoopDriver
MyWareWidgetNodeDriver
MyWareOtherNodeDriver
```

Now, if we register in our new driver called MyDriver:

```
# default registration
MyDriver.register(MyWareTypeNode, MyWareWidgetNoopDriver)
# specialized drivers
MyDriver.register(MyWareWidgetNode, MyWareWidgetNodeDriver)
MyDriver.register(MyWareOtherNode, MyWareOtherNodeDriver)
```

MyWareThingieNode will be driven by MyWareWidgetNoopDriver.

This kind of walk through inherances also works with more childs and classes