UD05: ArgoUML

1. Introduction.

In order to better understand the class diagrams and their notation, the following is a complete example. In addition, the ArgoUML modeling environment has been used to create these UML diagrams. This environment specializes in modeling with UML and other graphical languages. It also allows you to both generate code from diagrams and generate diagrams from source code.

The ArgoUML application can be used from Windows, Linux, and MacOS X operating systems. The exercise format will be used to provide step-by-step instructions on how to create a class diagram based on the requirements of a statement.

Example of invoice management

Invoices from a company's suppliers can be from service providers or from suppliers of products or materials. Each of the invoices, of either type, has in common:

- The invoice number.
- The date of the invoice.
- The total amount - which is calculated differently in some than in others.
- Customer data, which is the NIF and name, and must be found in a separate class.
- The detail of the invoice (whether it is materials or services).

For both service invoices and product or material invoices, the last invoice issued must be kept in order to keep the invoice number in mind.

There is a list of services with the description and price per hour of each.

The details of the invoices must include:

- In the invoices for services, each invoice contains a list of services with the date of service, the number of hours spent, and the price / hour per service. There may be more than one benefit from the same service on an invoice.
- Product invoices must include a description, unit price, and quantity for each product (each invoice must have at least one).

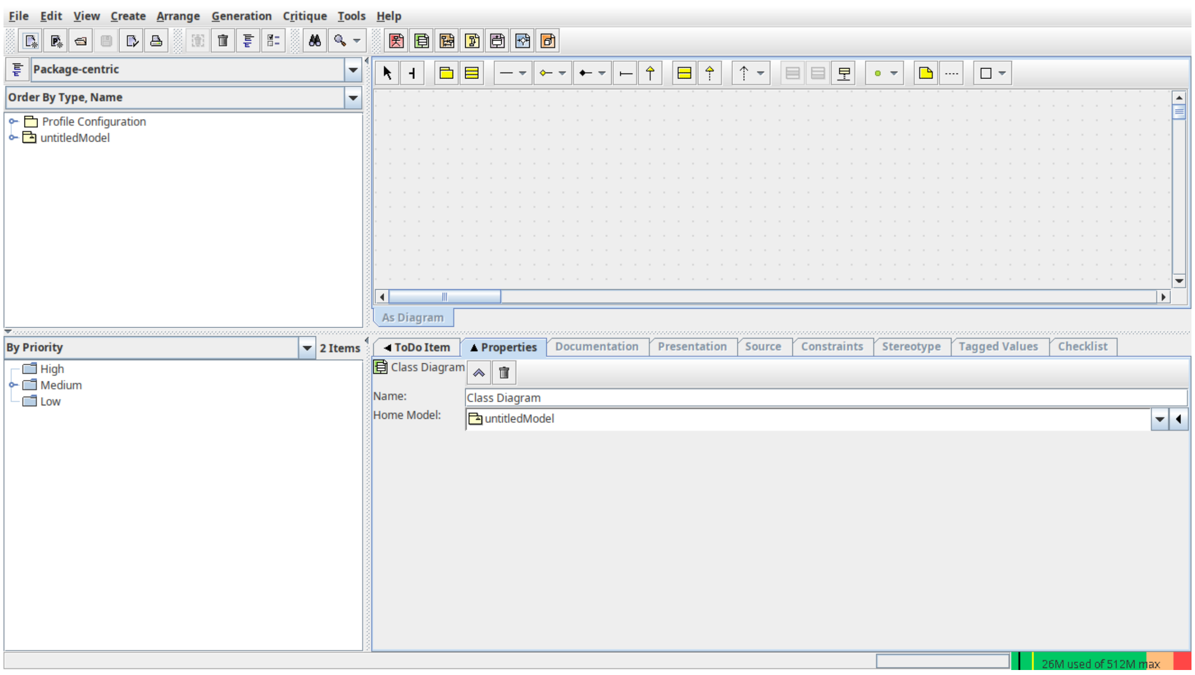
2. ArgoUML

The latest available version (0.35.1) can be found among the releases in the tigris github profile (https://github.com/argouml-tigris-org/argouml/releases).

We can download the zip file (https://github.com/argouml-tigris-org/argouml/releases/download/VERSION_0_35_1/ArgoUML-0.35.1.zip) and unzip it on our PC.

All releases are cross-platform, depending on whether we are on Linux, Windows, etc. we will launch the correct executable (sh, bat, etc...)

Once opened we will see something similar to this:

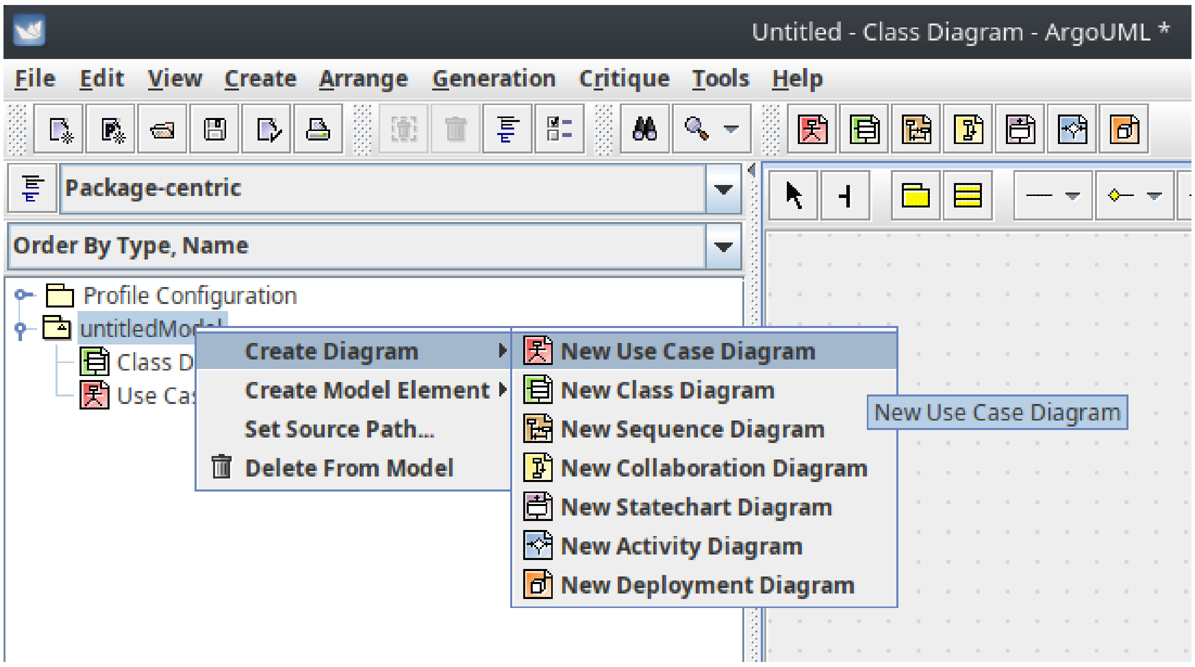


Name: Class Diagram
Home Model: untitledModel

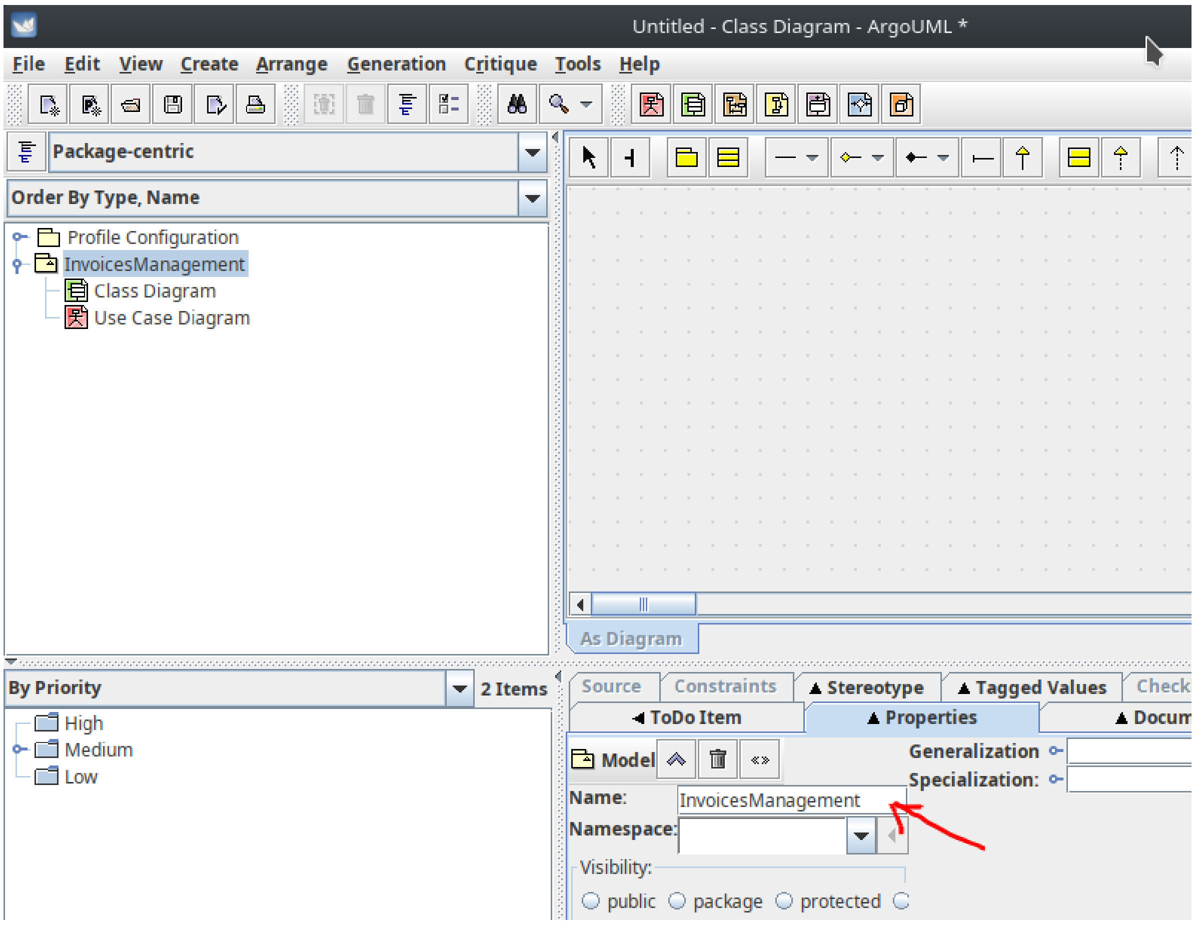
3. Our example step by step

3.1. Start ArgoUML

To start working with argoUML, it is only necessary to start the application, since it is entirely dedicated to the design of UML diagrams. By default it creates an empty untitledModel (that we can rename) with two empty diagrams (class and use case). By right-clicking on the model, we can choose to create more diagrams, like sequence diagrams...etc.

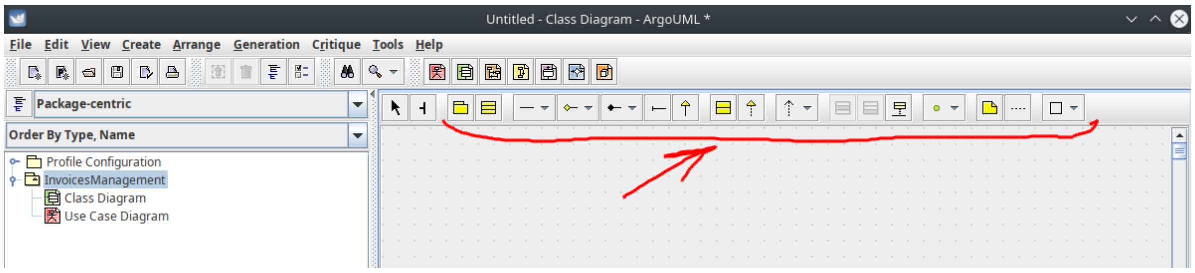


By now we only need to rename our model to InvoicesManagement:

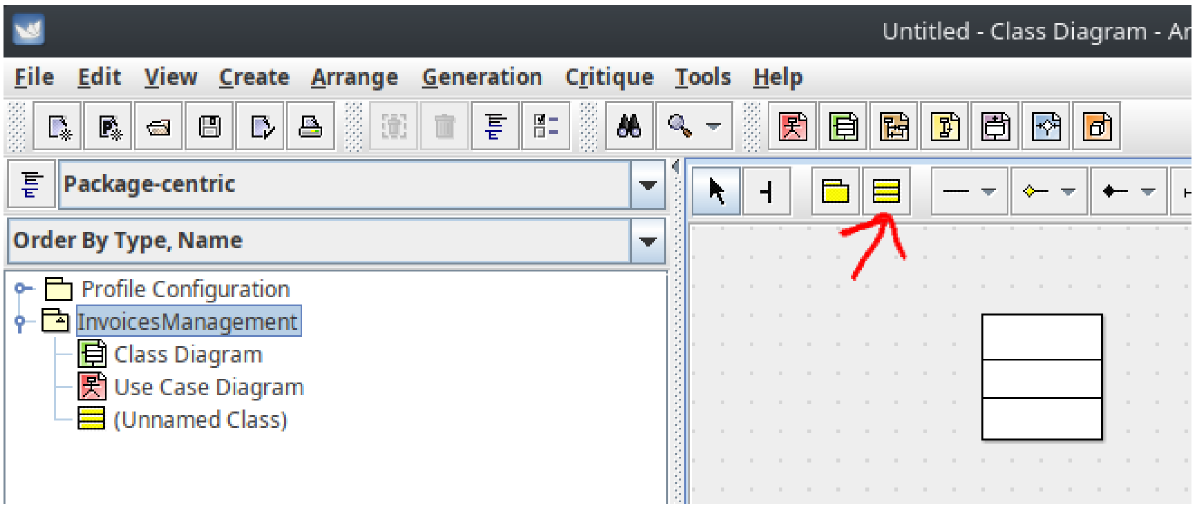


3.2. Basic ArgoUML

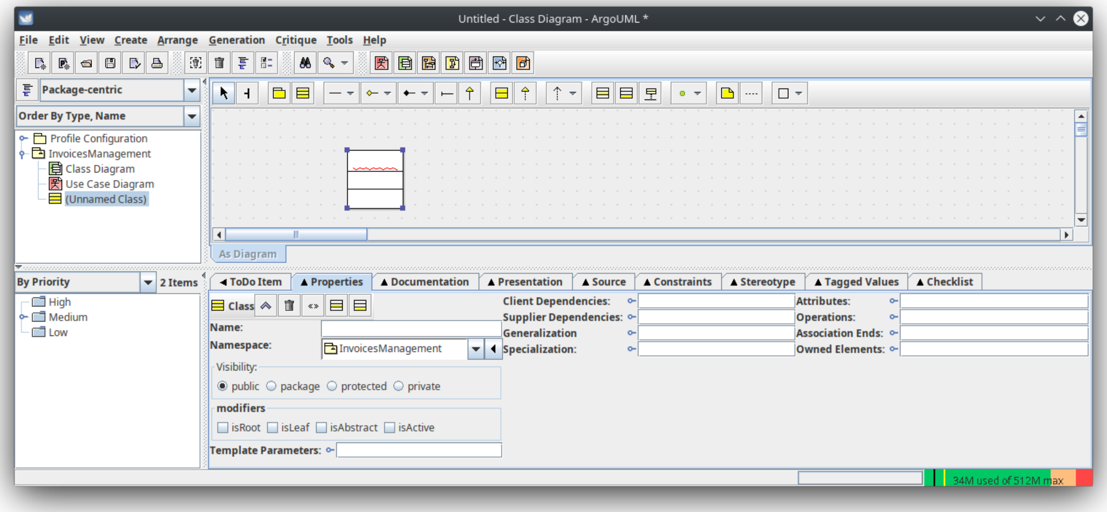
The start screen is actually very intuitive, containing a bar of entities that are identified by their own UML symbols.



To start drawing classes, click on the class box (the second in the palette), and mark some point in the drawing area. The class will appear, without any associated information.



In the lower right part of the window, the properties of the class appear, which can be modified to our liking at any time, just by selecting it. In the window to the left of the properties, the critics appear, which are recommendations that we should follow to obtain a good UML diagram.



Class

Name:

Namespace: InvoicesManagement

Visibility: [✓ public] package protected private

modifiers: isRoot isLeaf isAbstract isActive

Template Parameters:

Client Dependencies:

Supplier Dependencies:

Generalization:

Specialization:

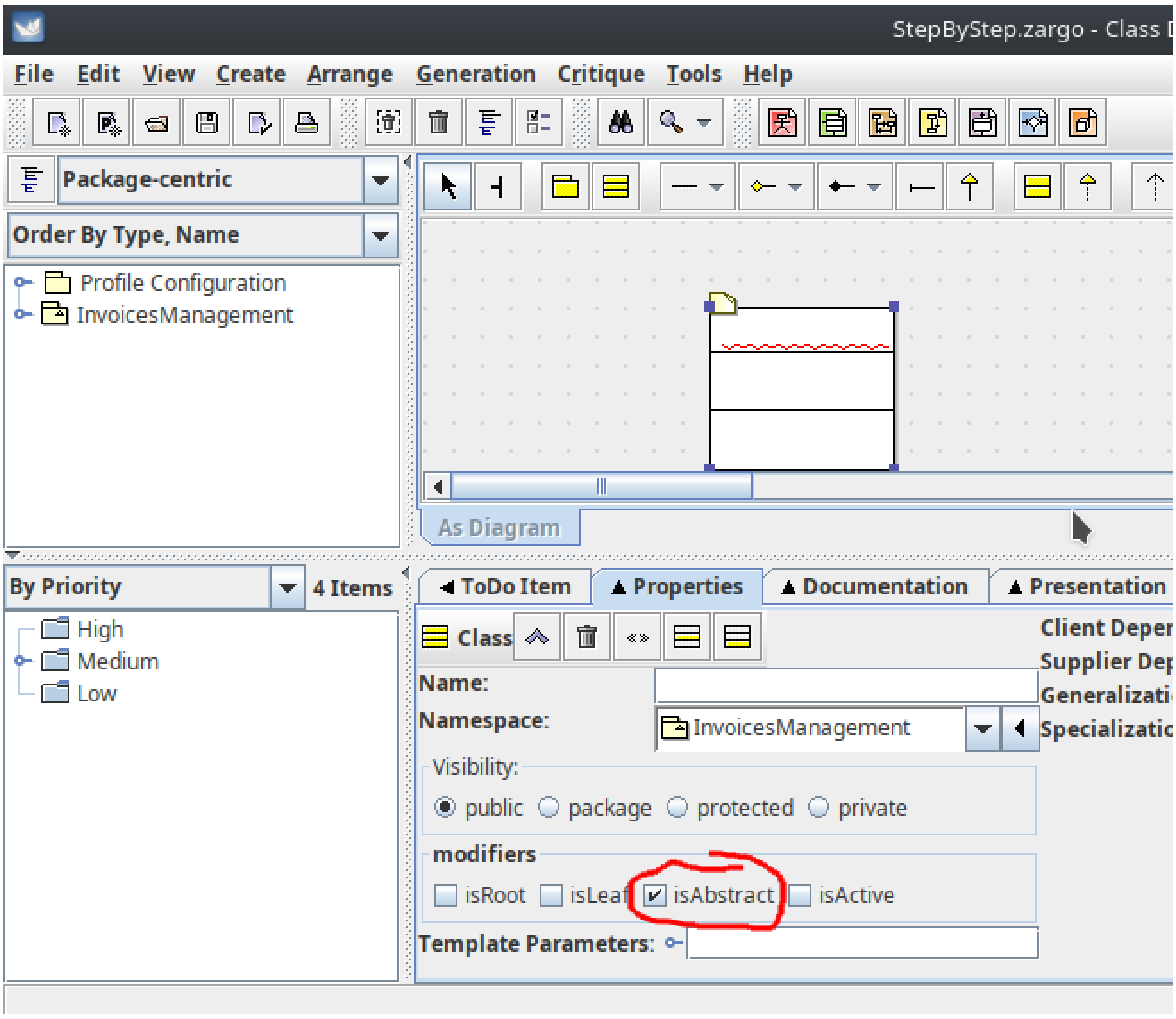
Attributes:

Operations:

Association Ends:

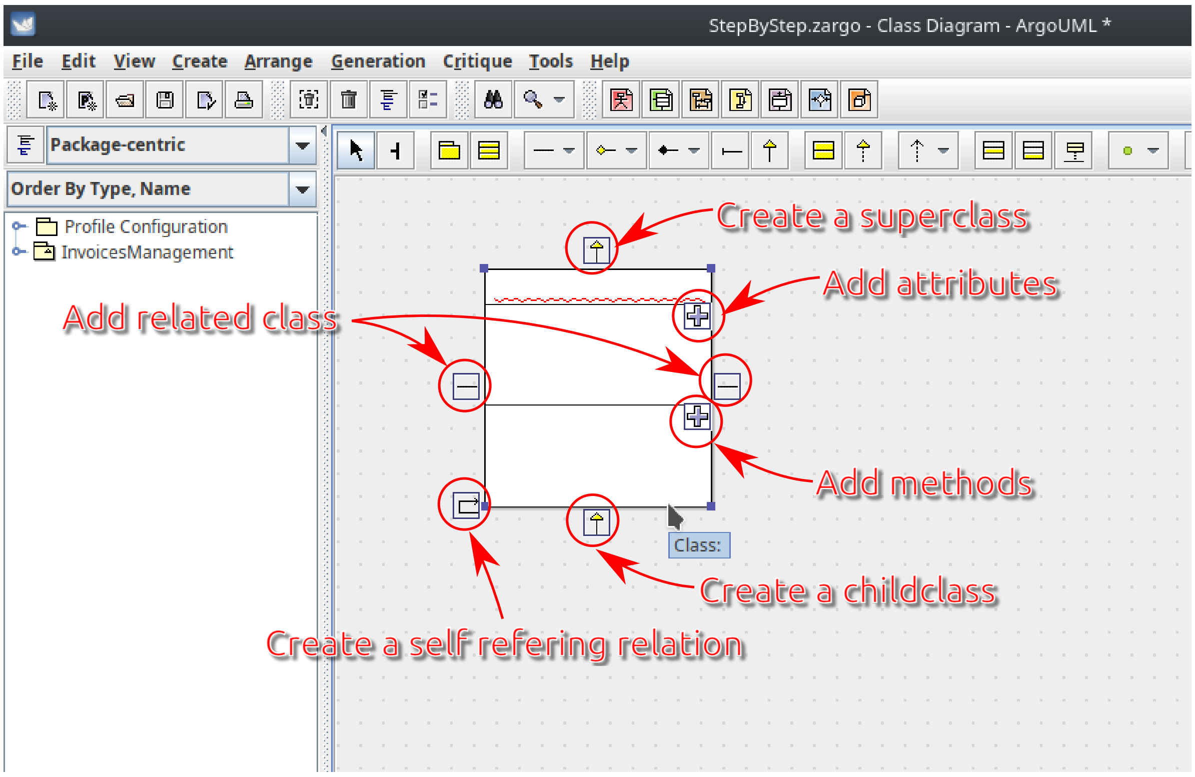
Owned Elements:

In the "attributes" and "operations" sections, we place everything (attributes and methods) related to the class. It is necessary to specify the name of the attribute or method, its visibility, its type, and its parameters (in the case of the method), separately. In this main tab, we mark the characteristics of the class itself, such as being abstract (as shown in the following figure).



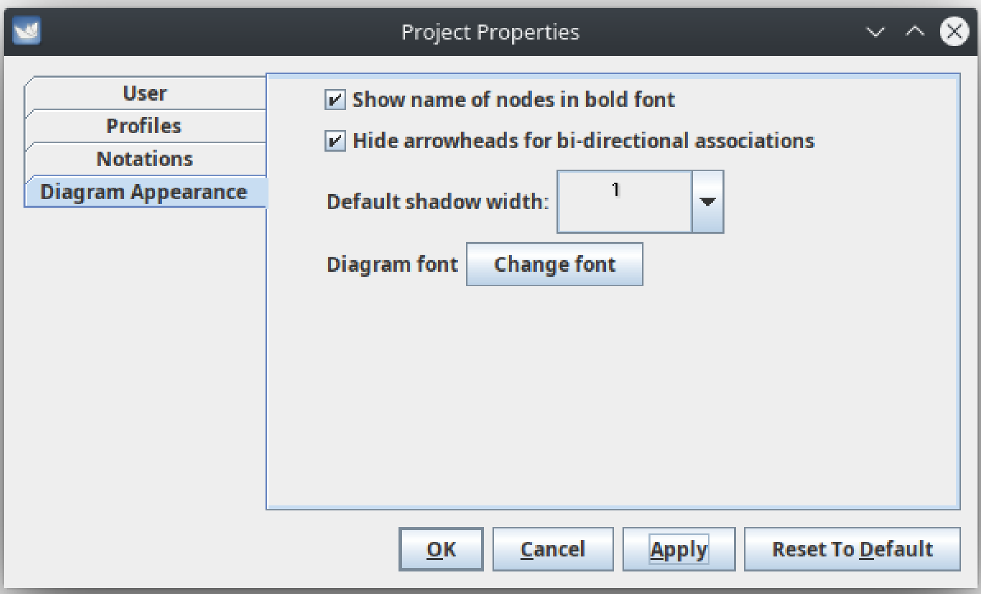
Name:
Namespace: InvoicesManagement
Visibility: [✓ public] package protected private
modifiers
isRoot isLeaf [✓ isAbstract] isActive
Template Parameters:

When a class is selected, a different kind of link could be used:



3.3. Recommended configuration

I recommend you to **show name of nodes in bold font** with `File/Project Properties/Diagram Appearance`:



Project Properties

User | Profiles | Notations | **Diagram Appearance**

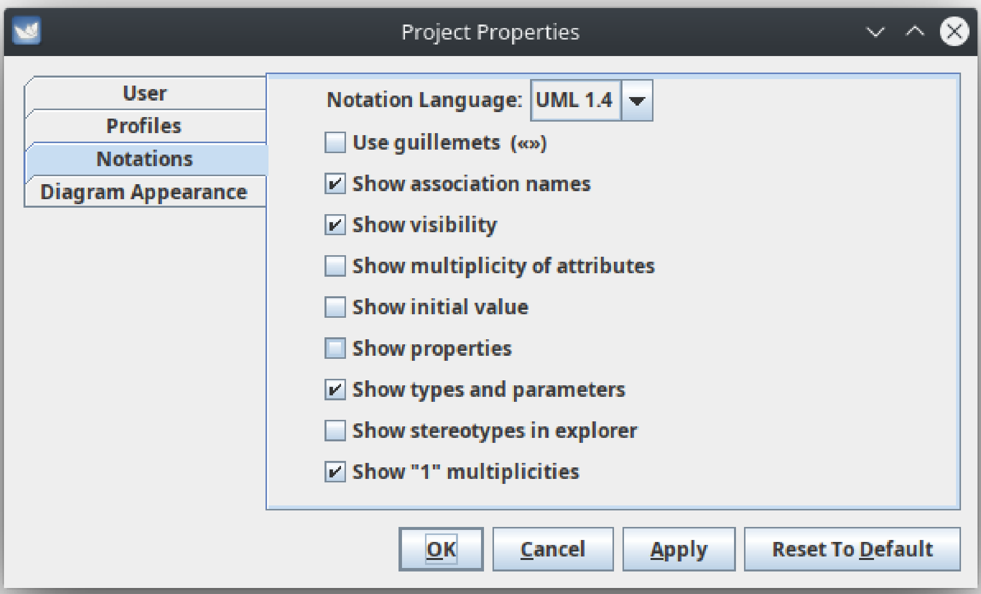
- [✓] Show name of nodes in bold font
- [✓] Hide arrowheads for bi-directional associations

Default shadow width: 1

Diagram font: Change font

OK | Cancel | Apply | Reset To Default

And to **show visibility** with `File/Project Properties/Notations`:



Project Properties

User | Profiles | **Notations** | Diagram Appearance

Notation Language: UML 1.4

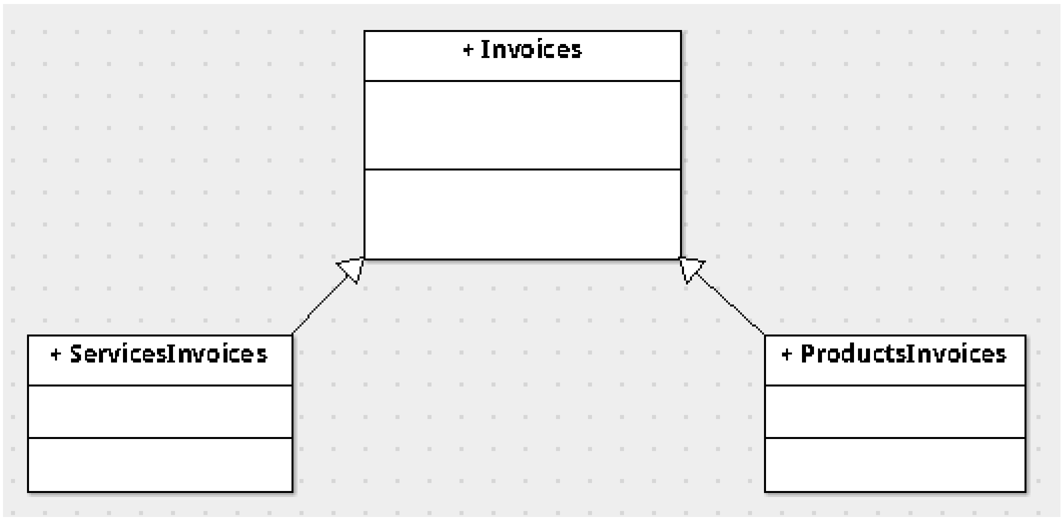
- [ ] Use guillemets («»)
- [✓] Show association names
- [✓] Show visibility
- [ ] Show multiplicity of attributes
- [ ] Show initial value
- [ ] Show properties
- [✓] Show types and parameters
- [ ] Show stereotypes in explorer
- [✓] Show "1" multiplicities

OK | Cancel | Apply | Reset To Default

3.4. The example step by step

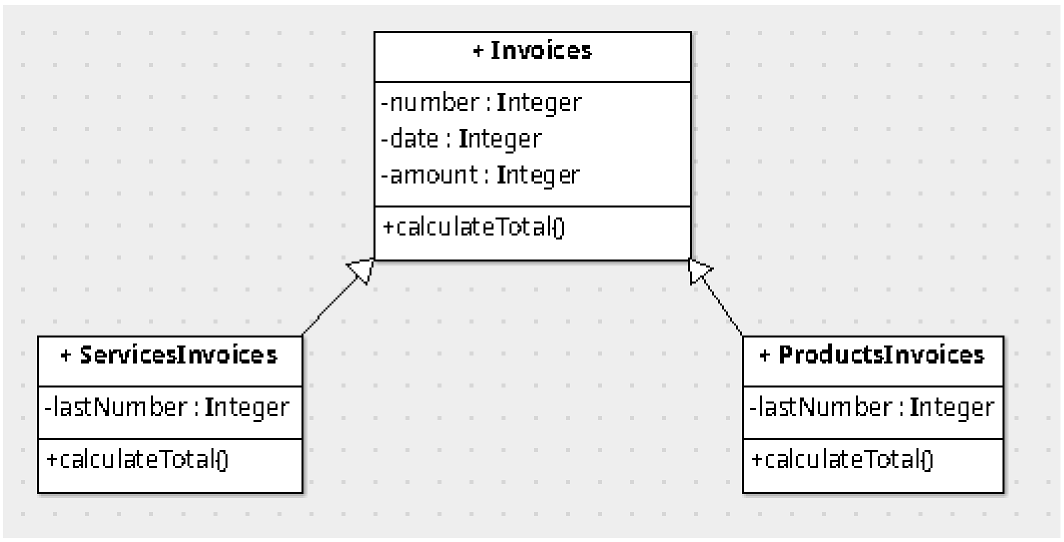
3.4.1. Create classes

Create three classes hierarchy, first create one of them (for example `Invoice` and after that create two childclasses: `ServicesInvoices` and `ProductsInvoices`). After that select each one and add the name in the Properties tab. Then you should see something like this:



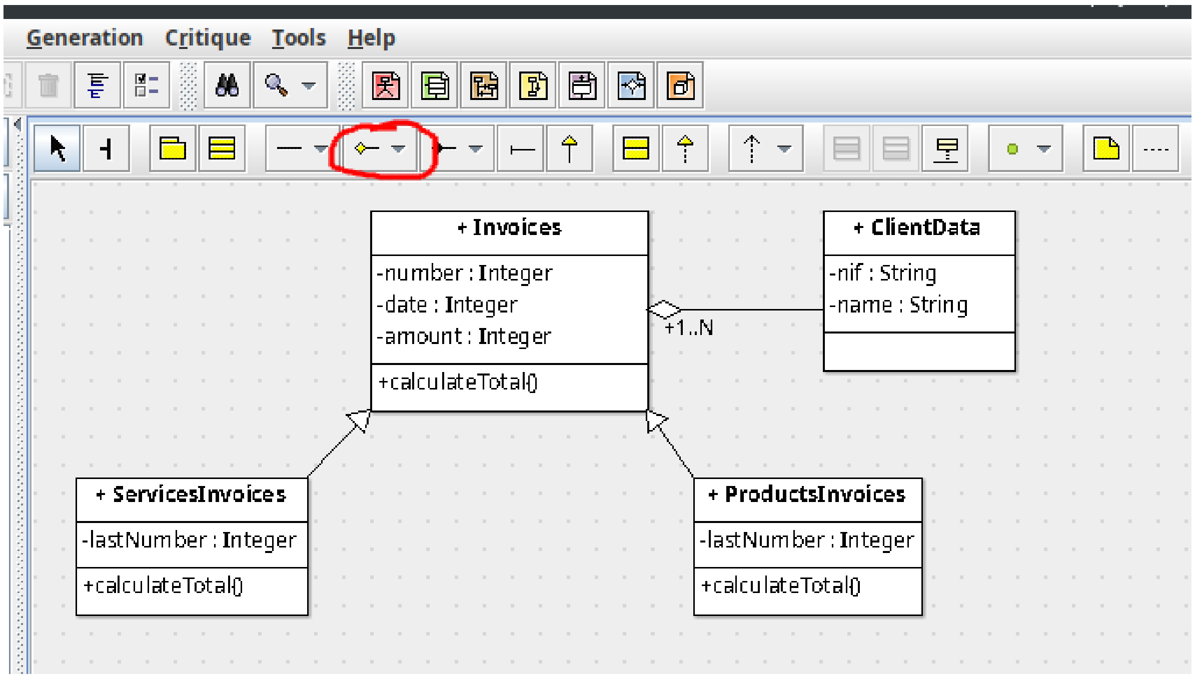
3.4.2. Add attributes and methods

Add attributes for each class, take care of selecting correct visibility for each attribute (by default private), and also select the correct data type. You get this:



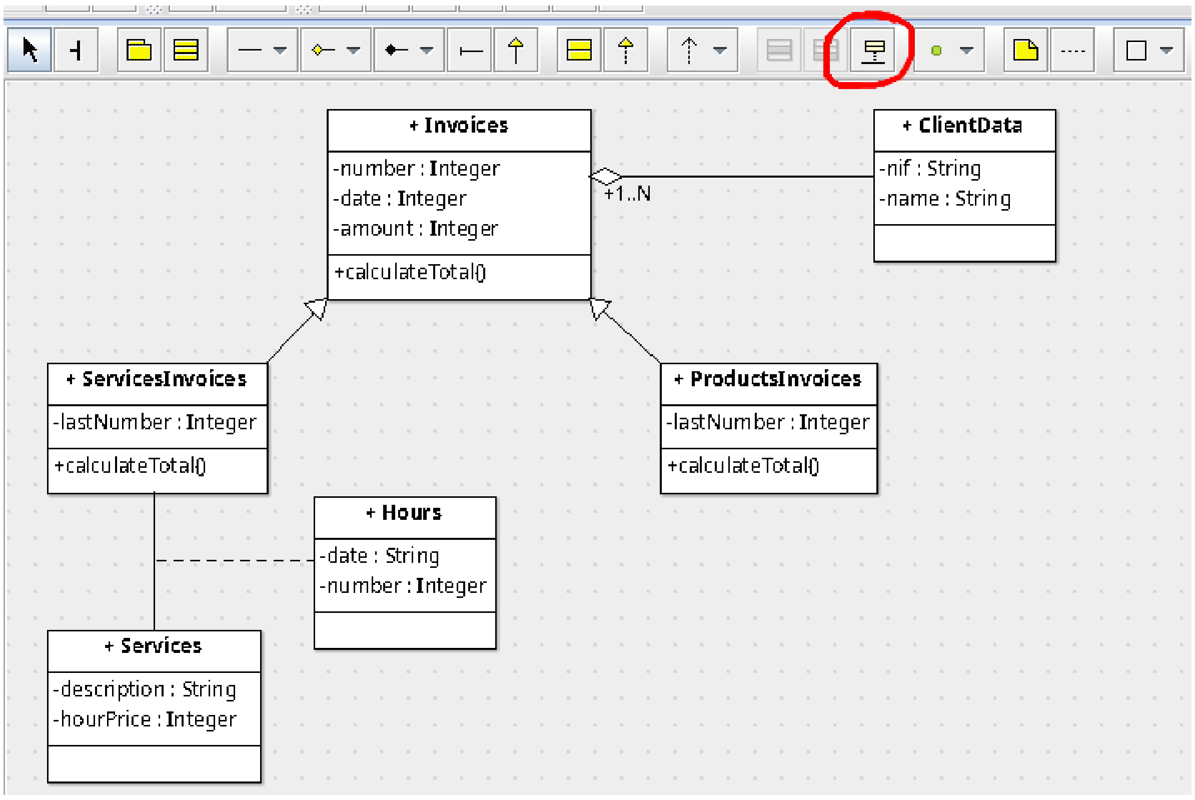
3.4.3. `ClientData` class

As said by the problem: "*Customer data, which is the NIF and name, and must be found in a separate class.*", so let's add as an Aggregation, because the client only has sense with an invoice. You need to select the correct button in the toolbar and then click and drag from one class (`Invoices`) to the other (`ClientData`)



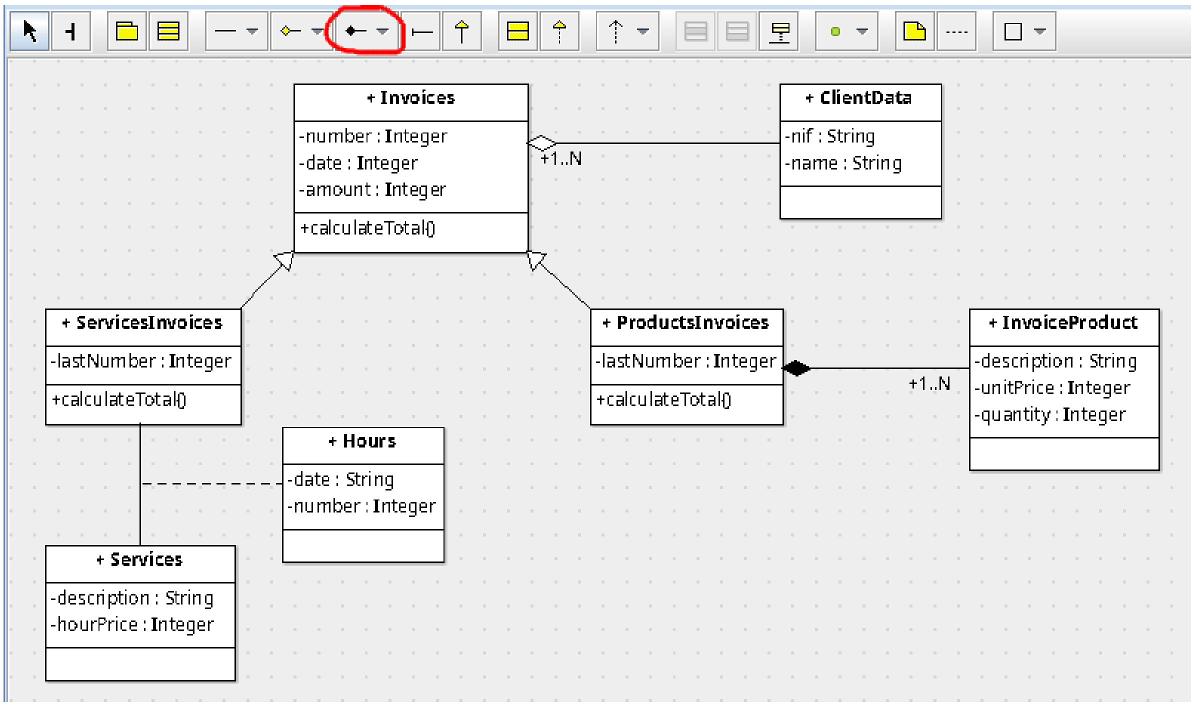
3.4.4. `Services` and `Hours` Class

From the line "*There is a list of services with the description and price per hour of each.*" we extract that the `Services` class has two attributes (`description` and `hourPrice`). And from "*...each invoice contains a list of services with the date of service, the number of hours spent, and the price / hour per service. There may be more than one benefit from the same service on an invoice.*" we need an associated `Hours` class with two attributes (`numberHours` and `description`):



3.4.5. `InvoiceProduct` Class

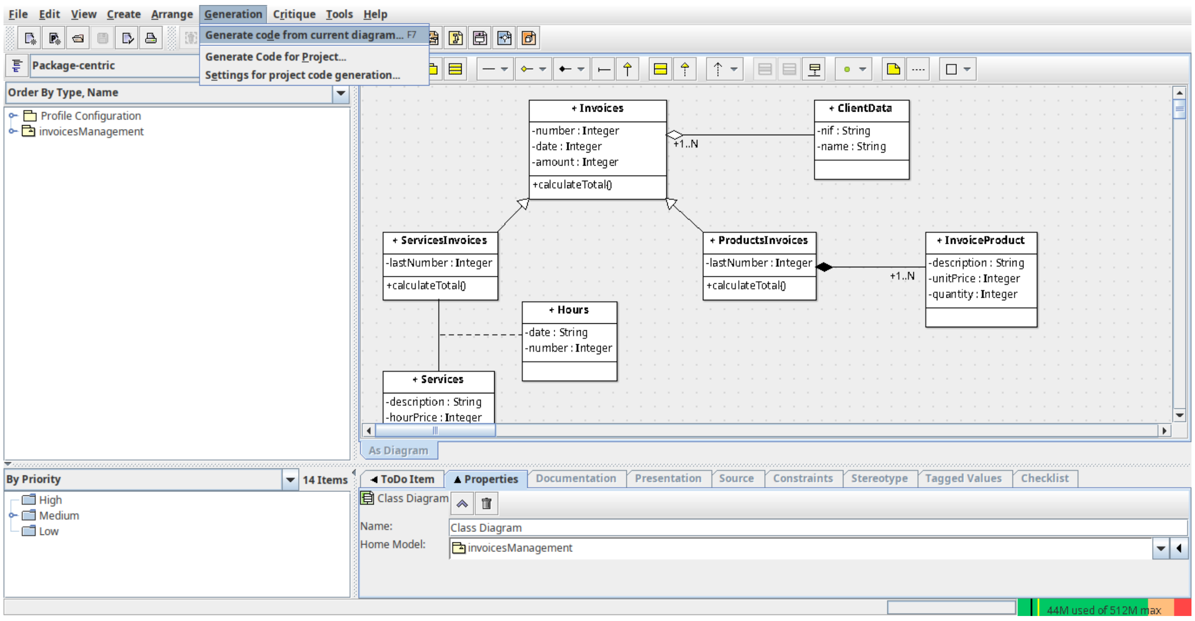
Last line says: "*Product invoices must include a description, unit price, and quantity for each product (each invoice must have at least one).*" so we need a `InvoiceProduct` class with three attributes (`description`, `unitPrice`, `quantity`) and the relation with `ProductsInvoices` class is a composition, be sure to select the correct button:


+ Invoices
-number : Integer
-date : Integer
-amount : Integer
+calculateTotal()
+1..N
+ ClientData
-nif : String
-name : String
+ ServicesInvoices
-lastNumber : Integer
+calculateTotal()
+ ProductsInvoices
-lastNumber : Integer
+calculateTotal()
+1..N
+ InvoiceProduct
-description : String
-unitPrice : Integer
-quantity : Integer
+ Hours
-date : String
-number : Integer
+ Services
-description : String
-hourPrice : Integer

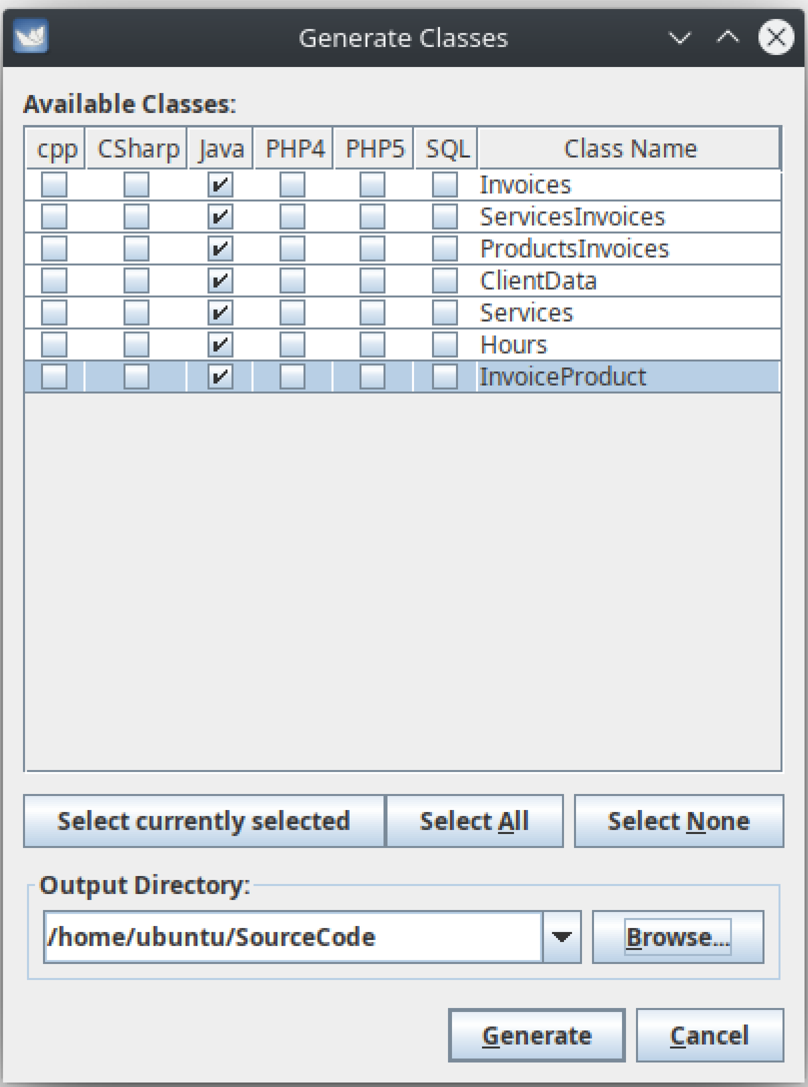
4. Generate source from ArgoUML project

Once we have our class Diagram in `ArgoUML` we can generate source code for `c` or `java` languages as described below:

Go to the menu `Generation/Generate Code from current Diagram` or press de `F7` key.



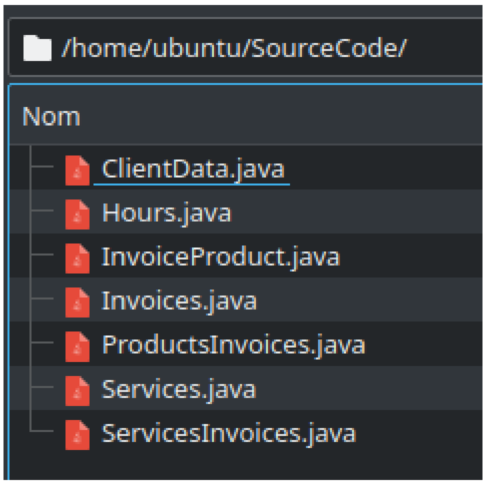
Then you must select destination language (c++, c#, java, PHP4, PHP5, SQL) and the classes you want to generate source code. You need also to specify the destination folder for the code:



cpp	CSharp	Java	PHP4	PHP5	SQL	Class Name
☐	☐	✓	☐	☐	☐	Invoices
☐	☐	✓	☐	☐	☐	ServicesInvoices
☐	☐	✓	☐	☐	☐	ProductsInvoices
☐	☐	✓	☐	☐	☐	ClientData
☐	☐	✓	☐	☐	☐	Services
☐	☐	✓	☐	☐	☐	Hours
☐	☐	✓	☐	☐	☐	InvoiceProduct

Output Directory: /home/ubuntu/SourceCode

Then just click `Generate` button and you get the folder with sources:



```
/home/ubuntu/SourceCode/
Nom
ClientData.java
Hours.java
InvoiceProduct.java
Invoices.java
ProductsInvoices.java
Services.java
ServicesInvoices.java
```

And if you open some of them you can see all the code automatically generated.

5. Exercises

5.1. Argo_01

Download `ArgoUML` for your Operating System and follow this step by step guide until point 3 (included) and send through AULES the `YOURNAME.zargo` file with this project.

AULES does not allow you to send *.zargo* file so, you have to compress to a zip file and send that file.

5.2. Argo_02

Complete the example on the youtube video from AULES about `Case Use Diagram` ("Empresa pedidos") completing it with the following instructions:

1. Draw the context diagram of the system.
2. Generate 3 case use description templates (different of the example).
3. Complete the final diagram.

Also, you have to add:

1. At least two hierarchy examples for actors.
2. A relationship with actors/systems outside our system.

Use your knowledge about real system to complete the case uses that are not visible on the youtube video.

Send through AULES the `YOURNAME.zargo` file with this project.

You could use this link as a guide: https://docplayer.es/90376253-Desarrollo-de-un-diagrama-de-casos-de-uso-mediante-el-software-argouml.html

5.3. Argo_03

Following point 4 of this guide generate the sources for the project in exercise `Argo_01` in `Java` and send the folder compressed in `zip` through AULES site.

5.4. Argo_04

Generate an UML class diagram from sources of the project that you choose from Unit05 of development (`gestionEmpleados`, `gestionHospital`, `contrarreloj`, `reservasLibreria`, `gestorCorreoElectronico`, `juegoDeCartas`). Explain how you do it with a memory in `pdf`, and send then `pdf` and `.zargo` file together in a zip through AULES.

6. Fuentes de información

- Apuntes IOC (Marcel García)
- Uso de *argoUML* para diseño de aplicaciones