# Tutorial: Code generation with xtend

Ansgar Radermacher (CEA LIST)

In this tutorial, we will write a simple code generator for Rust, supporting a subset of UML classes

## Pre-lab

**Objectives**:

- Install xtend SDK
- Import base generator plugin that is able to call the xtend generator
- Install Papyrus SW designer

**Exercises:**

1. Install the xtend SDK: Help -> Install New Software, select the update site corresponding to your Eclipse version (e.g. 2023-12), type “xtend” into the filter box. Select xtend SDK
2. Use the Papyrus SW designer update site
   (see https://wiki.eclipse.org/Papyrus/Designer/getting-started)
3. Import the project org.eclipse.papyrus.designer.rust.codegen available in the folder of this tutorial

## Lab 1 – Model library for Rust primitive types

**Objectives**: Write a model library for types used in Rust, enabling developers to references these types

**Exercises:**

1. Create a new UML model.
2. Create a couple of primitive types, as shown in the table below. Please note that we removed some variants of integer types to avoid reduce repetitive effort.

| | |
|---|---|
| i8 | The 8-bit signed integer type. see std::i8 module. |
| i32 | The 32-bit signed integer type. see std::i32 module. |
| u8 | The 8-bit unsigned integer type. see std::u8 module. |
| u32 | The 32-bit unsigned integer type. see std::u32 module. |
| usize | The pointer-sized unsigned integer type. see std::usize module. |
| f32 | The 32-bit floating point type. see std::f32 module. |
| f64 | The 64-bit floating point type. see std::f64 module. |
| String | A UTF-8–encoded, growable string, see std::string::String |

## Lab 2 – Write the code generator

**Objectives**:

1. Create a new UML model. This model should contain a class A with two attributes: i and str of type u32 and String, respectively
2. Write a generator that is able to produce Rust code for this UML class with suitable indentation. In the first variant, the generator should only support attributes. Classes can be mapped to structs in Rust, for the class A created above, the following code should be produced.

```
struct A {
   i: u32,
   str: String
}
```

## Lab 3 – Add operations

**Objectives**:

- Also map operations to Rust

**Exercises:**

1. Add an operation f to your class A. This operation should return an unsigned integer (u8).
2. Assure that the generation adds an *implementation block* with a constructor initializing the attributes and a loop over all operations. For simplicity, the generator does not have to produce the full list of parameters, but the type of the return parameter should be taken into account. Thus, the generator should produce the code below for the class A (in addition to the struct from the previous lab).

```
impl A {

    fn new(first: &str, name: &str) -> A {
        Person {
            first_name: first.to_string(),
            last_name: name.to_string()
        }
    }

    fn f(&self) -> u8 {
    }
}
```