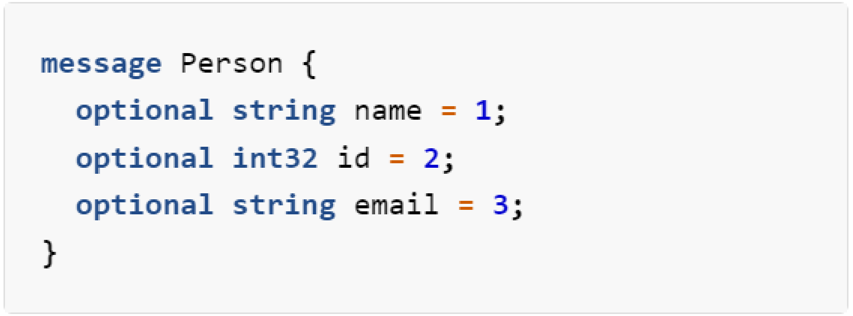
Rust Code Prototyping using XML

John Rexes Murro
Software Engineer at Amdocs

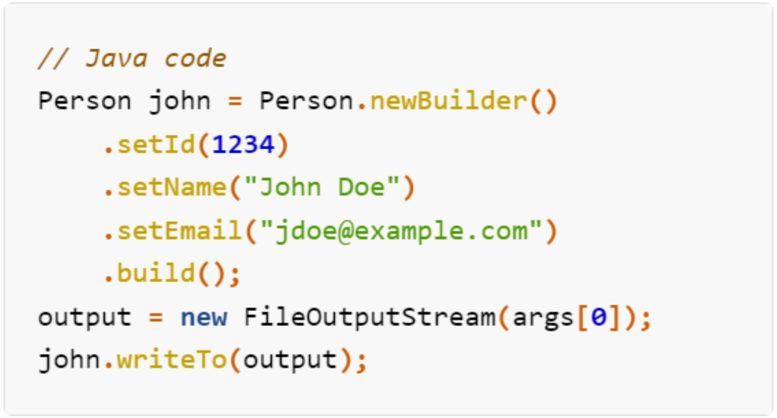
Protocol Buffers

Protocol buffers are Google's language-neutral, platform-neutral, extensible mechanism for serializing structured data – think XML, but smaller, faster, and simpler.



```
message Person {
  optional string name = 1;
  optional int32 id = 2;
  optional string email = 3;
}
```

A proto definition.



```
// Java code
Person john = Person.newBuilder()
    .setId(1234)
    .setName("John Doe")
    .setEmail("jdoe@example.com")
    .build();
output = new FileOutputStream(args[0]);
john.writeTo(output);
```

Using a generated class to persist data.

About rustxmlproto

OBJECTIVE

The objective is to create a library that can be used to generate Rustlang code from defining the structs, traits, enums and other objects by prototyping or modeling the object through XML.

XML Format

```xml
<prototype name="Foobar" class="struct" visibility="crate">
    <includes>
        <within name="foobar_within" scope="all/>
        <extern name="foobar_external" objects="Object1, Object2"/>
    </includes>
    <members>
        <String name="item" visibility="external"/>
        <u32 name="price"/>
    </members>
    <functions>
        <generic name="addDiscount">
            <parameters>
                <u32 name="discount"/>
            </parameters>
        </generic>
    </functions>
</prototype>
```

Rust module

```
import_proto!("foobar");

fn main() {
    let foobar_obj = Foobar::new(
      String::from("foobar"),
      123456789
    );
}
```

Main XML Elements

<prototype>

It is required to be the first or root element of the XML.

<procs>

Defines procedural or custom macros

<includes>

Defines all libraries that are to be imported

<members>

Defines all members in the object (specifically for structs and enums)

<functions>

Defines all functions or methods in the object

<prototype> attributes

```
<prototype name="Foobar" class="struct" visibility="crate">
```

name

Object name of the prototype

class

Defines the object type

visibility

Describes the visibility of the object

<includes> child elements

<within/>

Module is within the crate.

<extern/>

Module is from external crate.

```
<includes>
    <within name="foobar_within" scope="all/>
    <extern name="foobar_external" objects="Object1, Object2"/>
</includes>
```

<includes> child attributes

scope

Includes all objects within the defined scope

objects

Specifies the objects that are required to be used

```
<includes>
    <within name="foobar_within" scope="all/>
    <extern name="foobar_external" objects="Object1, Object2"/>
</includes>
```

<members> and <functions>

Child Element

Any string that refer to a datatype

```
<members>
    <String name="item" visibility="external"/>
    <u32 name="price"/>
</members>
```

<members> and <functions>

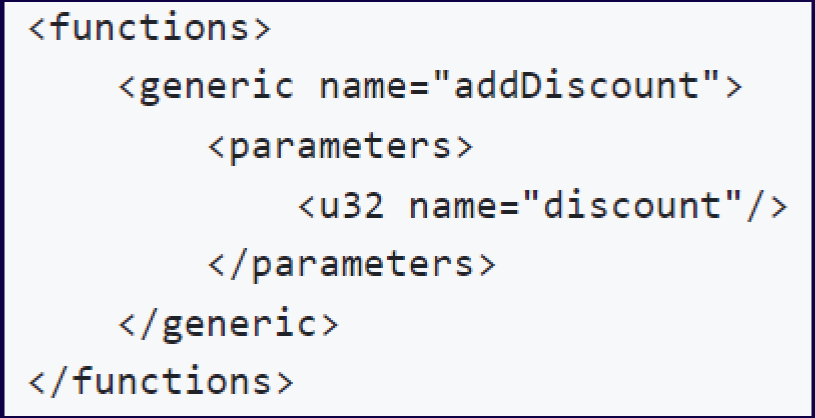
Child Element

Any string that refer to a datatype

Child Attribute

name – name of the member

visibility – visibility of the member

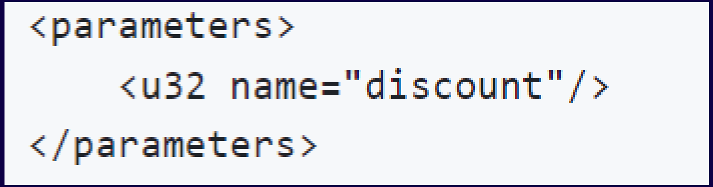


```
<functions>
    <generic name="addDiscount">
        <parameters>
            <u32 name="discount"/>
        </parameters>
    </generic>
</functions>
```

<parameters>

Child Element

Any string that refer to a datatype



```
<parameters>
    <u32 name="discount"/>
</parameters>
```

Child Attribute

name – name of the member

XML Elements for Macros

<derive>

Defines new inputs for the derive attribute

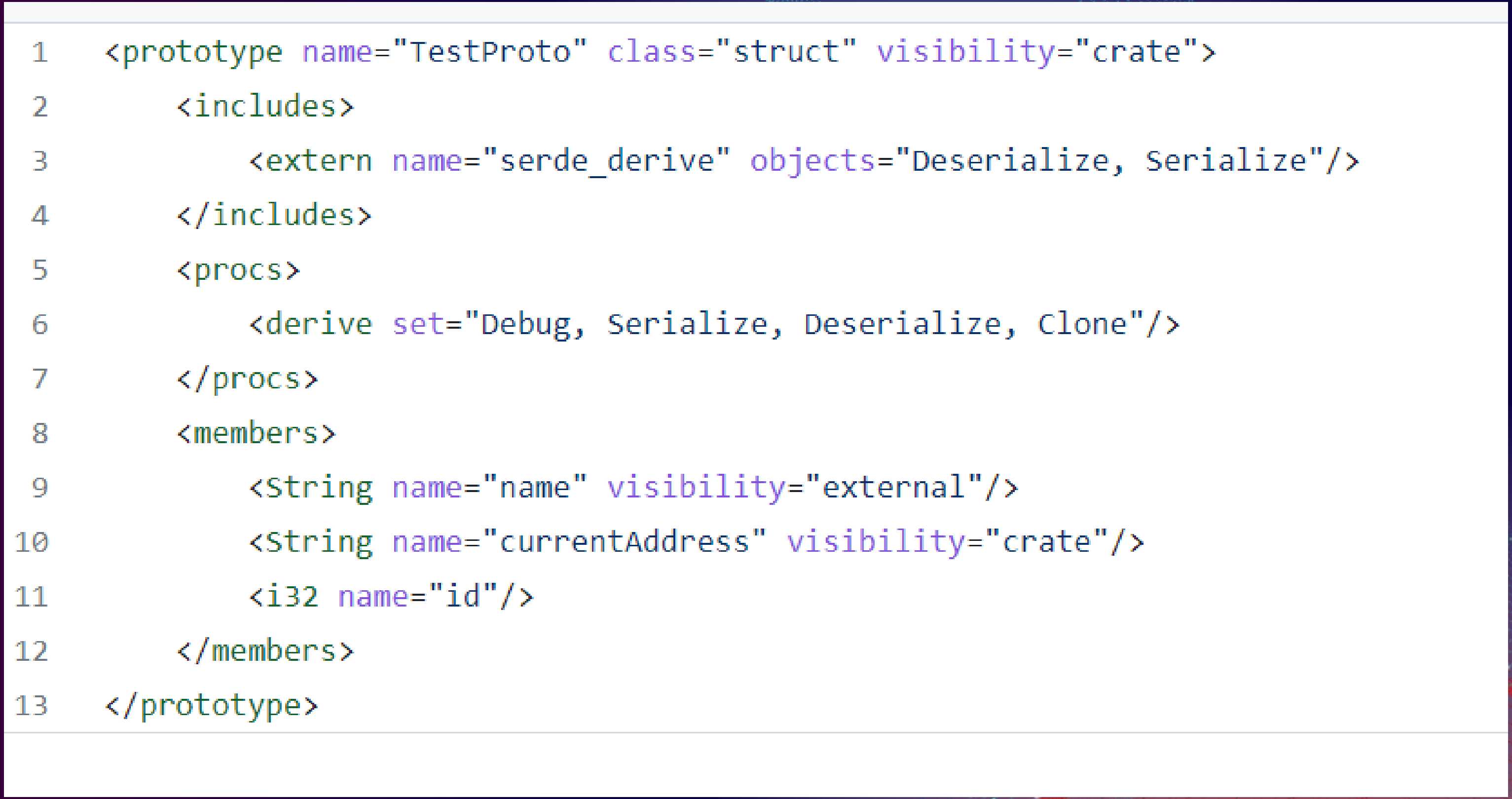
<custom>

Defines custom macros or other macros other than derive and serde

<serde>

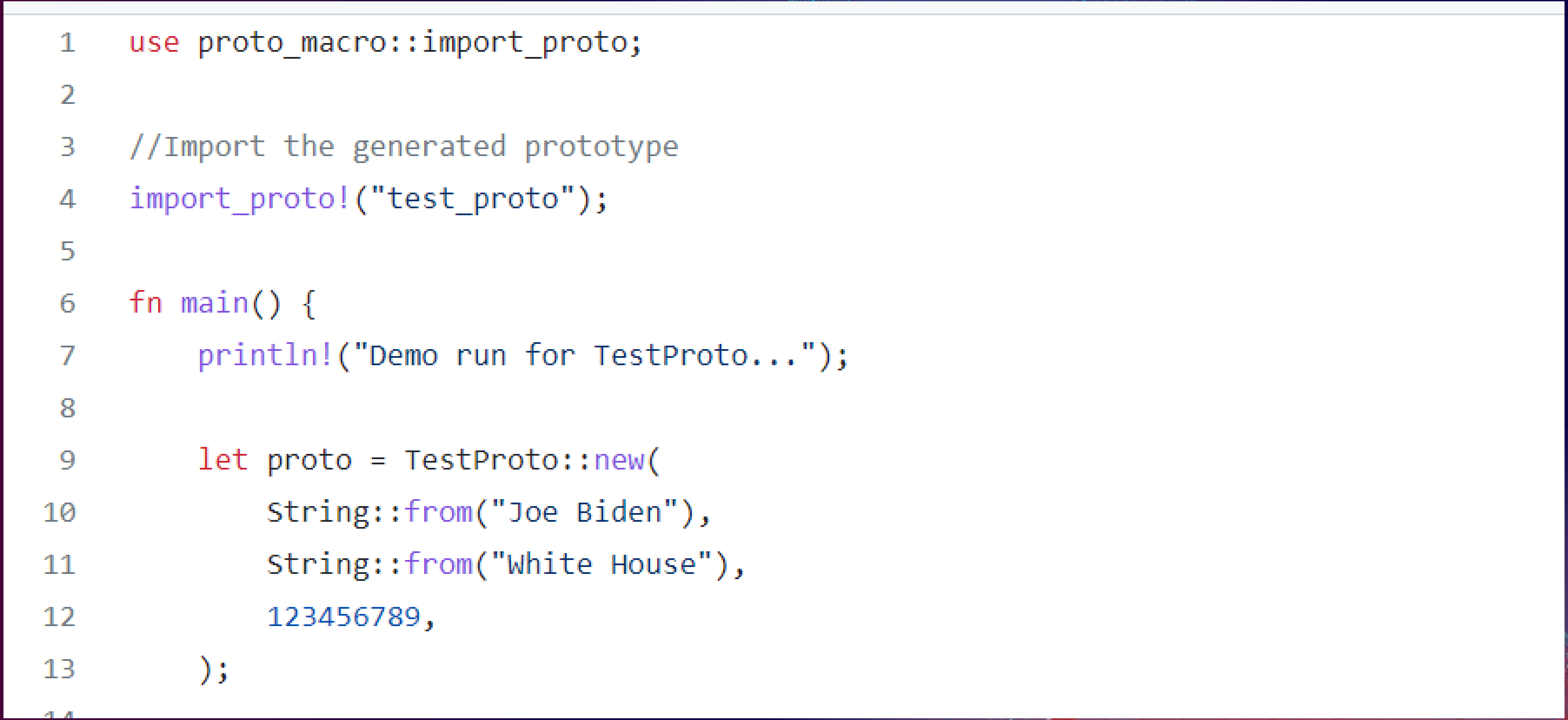
Defines the derive macro to generate implementations of Serialize and Deserialize traits

Demo Project – TestProto.xml



```xml
<prototype name="TestProto" class="struct" visibility="crate">
    <includes>
        <extern name="serde_derive" objects="Deserialize, Serialize"/>
    </includes>
    <procs>
        <derive set="Debug, Serialize, Deserialize, Clone"/>
    </procs>
    <members>
        <String name="name" visibility="external"/>
        <String name="currentAddress" visibility="crate"/>
        <i32 name="id"/>
    </members>
</prototype>
```

Demo Project – main.rs



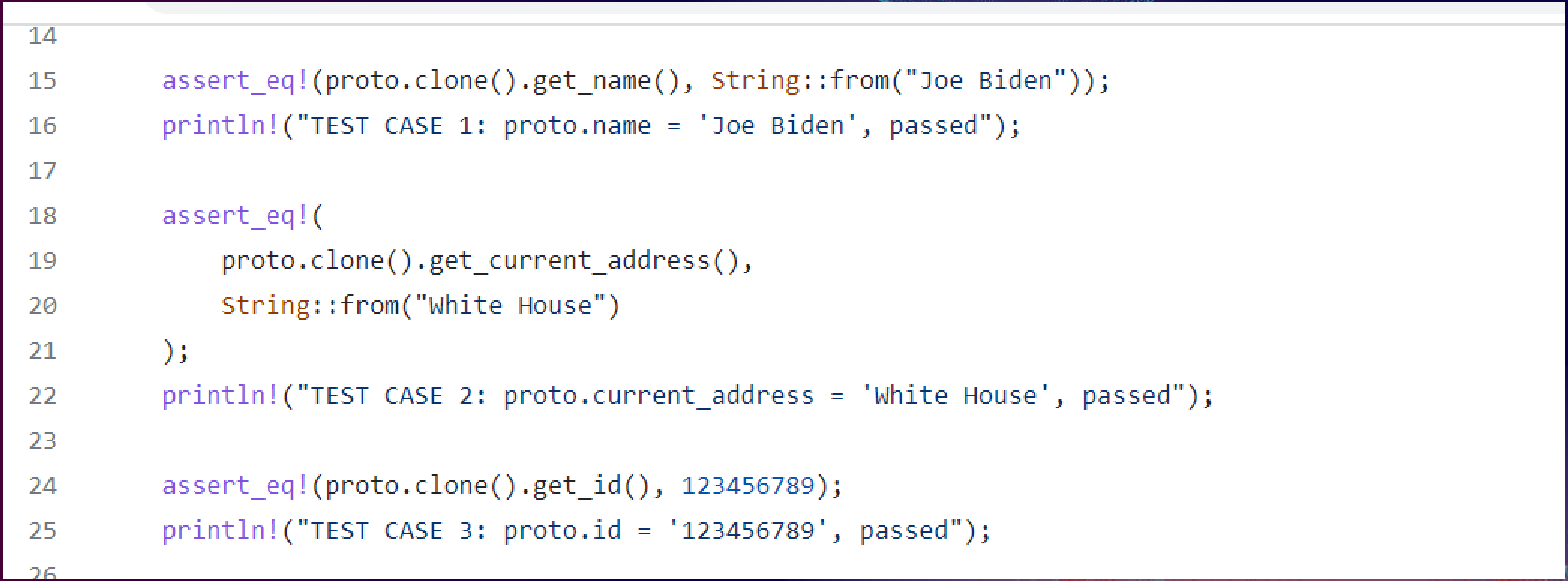
```rust
use proto_macro::import_proto;

//Import the generated prototype
import_proto!("test_proto");

fn main() {
    println!("Demo run for TestProto...");

    let proto = TestProto::new(
        String::from("Joe Biden"),
        String::from("White House"),
        123456789,
    );
```

Demo Project – main.rs



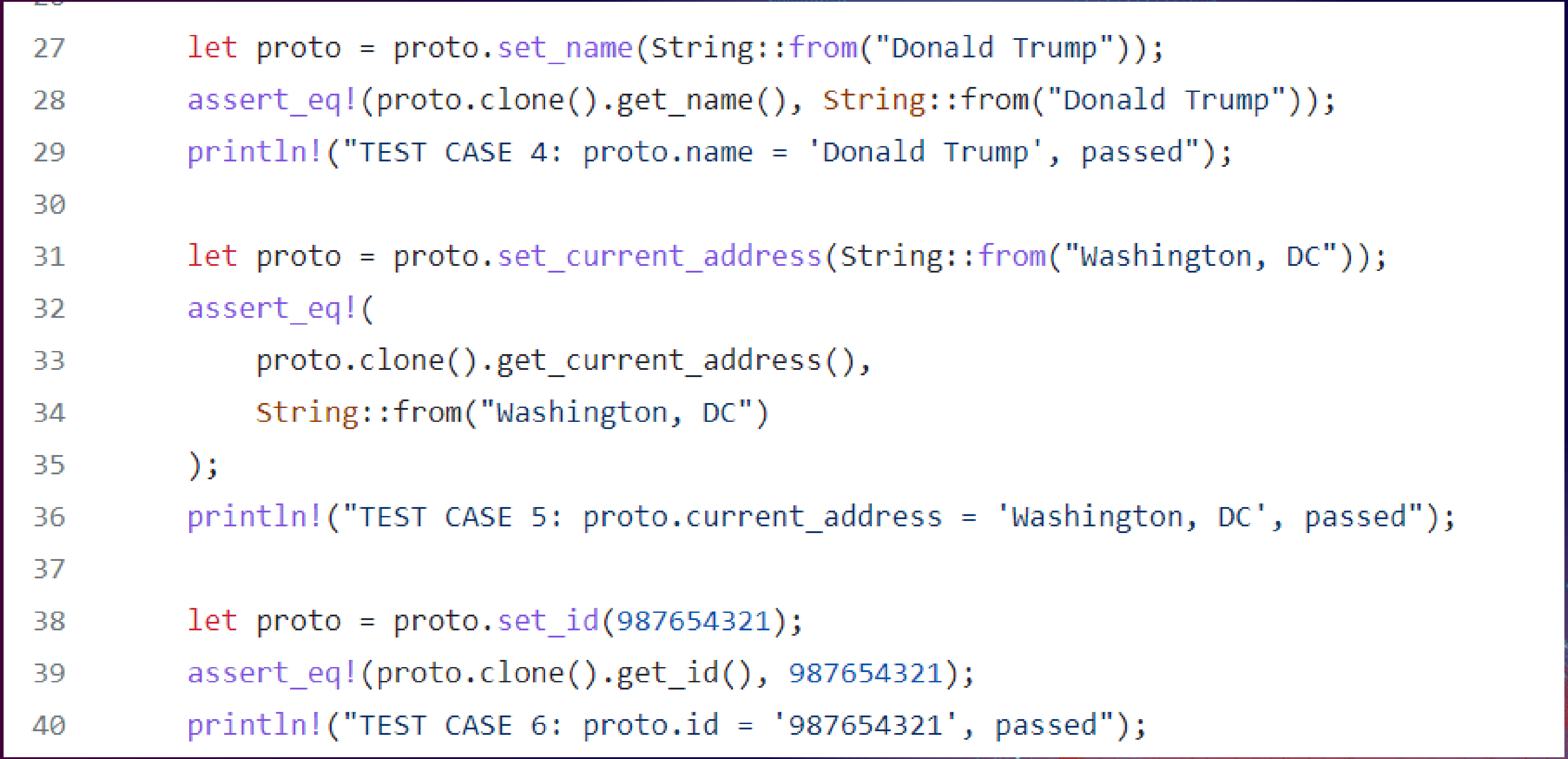
```

    assert_eq!(proto.clone().get_name(), String::from("Joe Biden"));
    println!("TEST CASE 1: proto.name = 'Joe Biden', passed");

    assert_eq!(
        proto.clone().get_current_address(),
        String::from("White House")
    );
    println!("TEST CASE 2: proto.current_address = 'White House', passed");

    assert_eq!(proto.clone().get_id(), 123456789);
    println!("TEST CASE 3: proto.id = '123456789', passed");
```

Demo Project – main.rs



```rust
    let proto = proto.set_name(String::from("Donald Trump"));
    assert_eq!(proto.clone().get_name(), String::from("Donald Trump"));
    println!("TEST CASE 4: proto.name = 'Donald Trump', passed");

    let proto = proto.set_current_address(String::from("Washington, DC"));
    assert_eq!(
        proto.clone().get_current_address(),
        String::from("Washington, DC")
    );
    println!("TEST CASE 5: proto.current_address = 'Washington, DC', passed");

    let proto = proto.set_id(987654321);
    assert_eq!(proto.clone().get_id(), 987654321);
    println!("TEST CASE 6: proto.id = '987654321', passed");
```

THANKS!

Do you have any questions?

jrmurro@gmail.com
+63 9310003044
https://linkedin.com/in/rexes-murro