Solidity contracts resemble a lot to Java objects. As in Java, we will be coding the "**template**" of a Smart Contract (what would be called the class) and we will be creating instances of said "**template**" by deploying the Smart Contract.

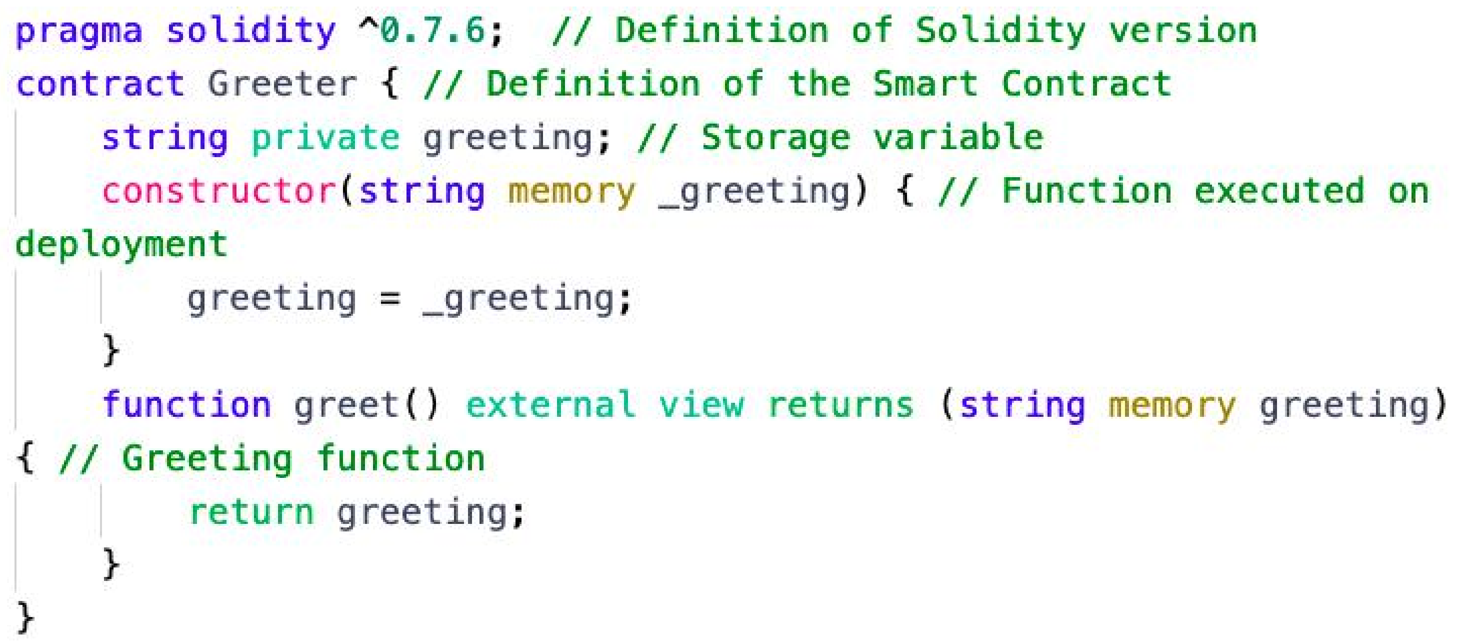
- ✓ Note that we will be referring to either the "template" of the Smart Contract or an instance as Smart Contract (as there are no special words for any of the two options) and the context should determine which one we are talking about.
- ✓ Also note that we will abbreviate Smart Contract as SC or contract.

Smart Contracts are made of mainly two different components:

1. **Storage variables(also called state variables):** They will be part of the state of each contract, meaning that they persist after a transaction executed, they are unique per instance and they won't be shared. Analogous to properties in Java.

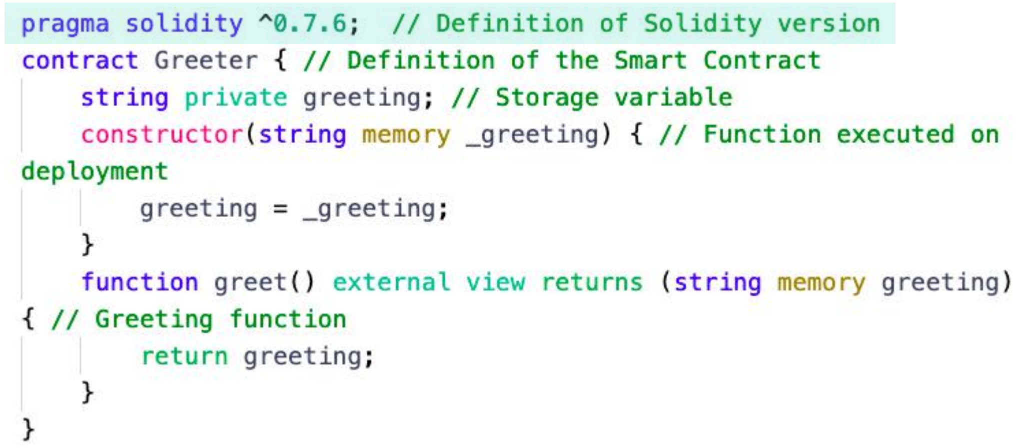
2. **Functions:** This will be the code of our contracts. Analogous to methods in Java. As in Java, there are special functions like the constructor for example.

Solidity Track
Solidity - Our first contract!



```
pragma solidity ^0.7.6;  // Definition of Solidity version
contract Greeter { // Definition of the Smart Contract
    string private greeting; // Storage variable
    constructor(string memory _greeting) { // Function executed on
deployment
        greeting = _greeting;
    }
    function greet() external view returns (string memory greeting)
{ // Greeting function
        return greeting;
    }
}
```

Let’s break it up!



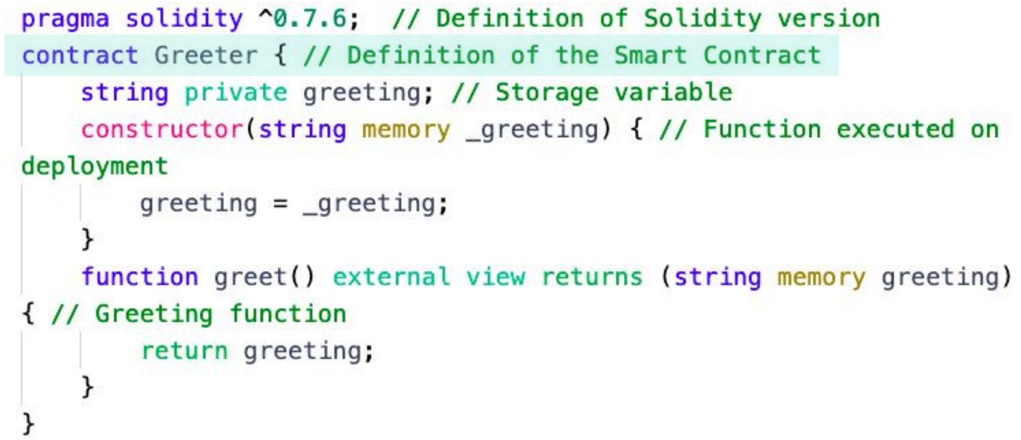
```
pragma solidity ^0.7.6;  // Definition of Solidity version
contract Greeter { // Definition of the Smart Contract
    string private greeting; // Storage variable
    constructor(string memory _greeting) { // Function executed on
deployment
        greeting = _greeting;
    }
    function greet() external view returns (string memory greeting)
{ // Greeting function
        return greeting;
    }
}
```

This defines the Solidity version that we are using, which is used by the compiler as a safety check.

This statement means that if we try to compile with a version that it is not equal or greater than 0.7.6, and lesser than 0.8.0, the compiler will throw an error.

(Optional) For more information check the official docs about this.

Let's break it up!



```
pragma solidity ^0.7.6;  // Definition of Solidity version
contract Greeter { // Definition of the Smart Contract
    string private greeting; // Storage variable
    constructor(string memory _greeting) { // Function executed on
deployment
        greeting = _greeting;
    }
    function greet() external view returns (string memory greeting)
{ // Greeting function
        return greeting;
    }
}
```

In this statement we define the name of our Smart Contract. This line will become more interesting once we learn about inheritance.

Let's break it up!

```
pragma solidity ^0.7.6;  // Definition of Solidity version
contract Greeter { // Definition of the Smart Contract
    string private greeting; // Storage variable
    constructor(string memory _greeting) { // Function executed on
deployment
        greeting = _greeting;
    }
    function greet() external view returns (string memory greeting)
{ // Greeting function
        return greeting;
    }
}
```

Here we define our first storage variable(or state variables). First we define its **type**, which in this case is a **string**, and after that its **visibility** which in this case is **private.**

Most common "atomic" types are:

- bool: can be used to represent scenarios that have binary results, such as true

  or false

- uint256: Unsigned integer of 256 bits(there
- address: 20 bytes(40 hex characters) that represent a Smart Contract or an EOA

- bytes32: 32 raw bytes

- bytes: Dynamic length raw bytes

- string: Dynamic length string

More types can be found in:

https://docs.soliditylang.org/en/develop/types.html#value-types

Important note:

Solidity does NOT support floating point numbers, to use numbers with decimal part you will have to use fixed point arithmetic which is supported natively in solidity 0.8.0 or greater (to use it in previous versions, you will have to handle them manually, this will be explained later)

Possible options for the visibility of storage variables:

Private:
Can be read and modified only by itself

Internal:
Can be read and modified by the Smart Contract and its inheritors(cannot be read or modified by other Smart Contract, even if the other contracts belong to the same "class")

Public:
Can be read by the Smart Contract, its inheritors, and other smart contracts(can be modified by its inheritors or itself, not by other contracts)