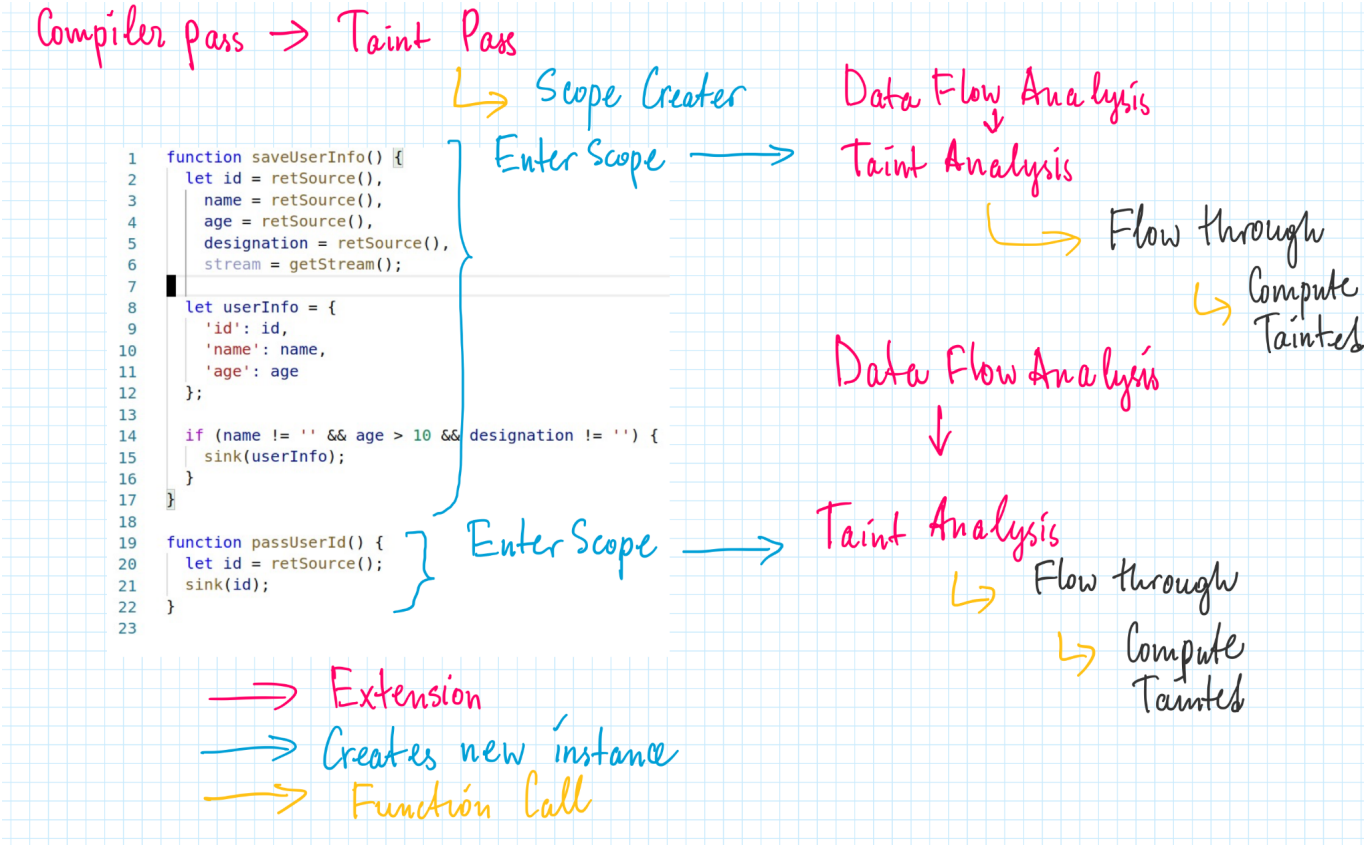
Intra-procedural Static Taint Analysis of JavaScript Programs

How it works

1. I've created class named **TaintPass.java** extends from **CompilerPass** as **Custom Compiler Pass**.
2. In the **TaintPass**, I have used **ScopeCreator** class to analyze for every function. For example, if a source code contains multiple functions, TaintPass will create multiple **analizer**.
3. **TaintAnalysis** is the main analyzer extends from **DataFlowAnalysis** with **forward** direction.
4. I've written **computeTainted** method to calculate **sources** may or must reach to sinks.

The following picture is aim to explain as a whole:



The example follows and how it works in **Control Flow Graph**:

```
function saveUserInfo() {
 let id = retSource(),
   name = retSource(),
   age = retSource(),
   designation = retSource(),
   stream = getStream();

 let userInfo = {
    'id': id,
    'name': name,
    'age': age
```

```
12  };
13
14  if (name != '' && age > 10 && designation != '') {
15    sink(userInfo);
16  }
17 }
```

Line number	Node type	Value	Parent	Sources may reach	Sources must reach
1	FUNCTION	saveUserInfo		{}	{}
2	VAR	id		{}	{}
3	VAR	name		{}	{}
4	VAR	age		{}	{}
5	VAR	designation		{}	{}
6	VAR	stream		{}	{}
7	EMPTY			{}	{}
8	VAR	userInfo	{id, name, age}	{}	{}
	OBJECTLIT	id, name, age		{}	{}
14	IF	{conditional:true}		{}	{}
15	EXPR_RESULT	sink userInfo	{id, name, age}	{id, name, age}	