Homework 3 New

Classification of Java Errors

1. **x+++-y**
   - Not a compile time error
2. **x---+y**
   - Not a compile time error
3. **Incrementing a read-only variable**
   - Semantic error
4. **Code in class C accessing a private field from class D**
   - Semantic error
5. **Using an uninitialized variable**
   - Semantic error
6. **Dereferencing a null reference**
   - Not a compile time error
7. **null instanceof C**
   - Not a compile time error
8. **!!x**
   - Not a compile time error
9. **x > y > z**
   - Semantic error
10. **if (a instanceof Dog d) {...}**
    - Not a compile time error
11. **var s = """This is weird""";**
    - Syntax error
12. **switch = 200;**
    - Syntax error
13. **x = switch (e) {case 1->5; default->8;};**
    - Not a compile time error

How do JavaScript and Rust treat the following:
let x = 3;
let x = 3;

- In Rust, the first variable is declared and the second variable shadows the first, and gives a warning. In JS, the let keyword designates an mutable, reassignable variable optionally initializing to a value (3 in this case), and here it errors in JavaScript saying that x has already been declared. (Syntax error)

Describe how the languages Java and Ruby differ in their interpretations of the meaning of the keyword private

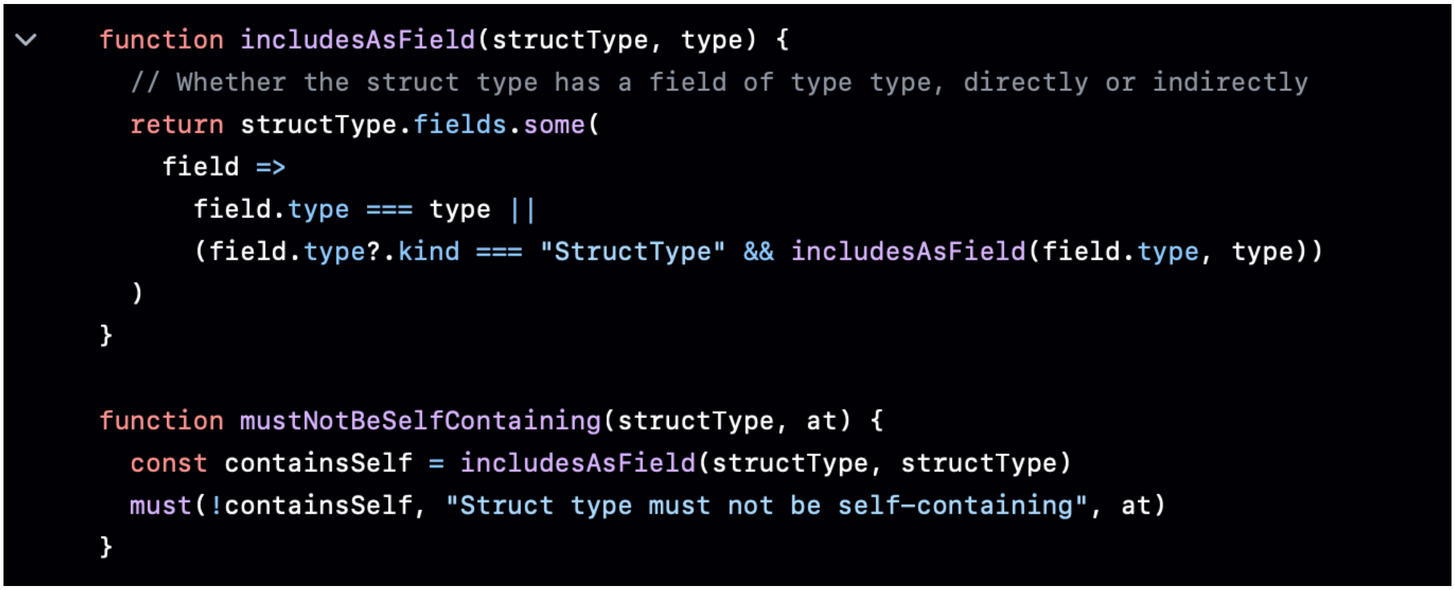
- In Java, the private keyword is an access modifier used for attributes, methods and constructors making them only able to be used within their declared classes. They are also enforced at compile-time. In Ruby, private is enforced at runtime and is instance based rather than class based.

Some languages do not require the parameters to a function call to be evaluated in any particular order. Is it possible that different evaluation orders can lead to different arguments being passed? If so, give an example to illustrate this point, and if not, prove that no such event could occur.

- This is definitely possible to do, if a language evaluates an arithmetic operation from left to right instead of from right to left, it can change the value of the result depending on which the language decides to use. Imagine you were incrementing the same variable on a function call with two numbers and you called f(x++, x++) the value of x would be different depending on which side is evaluated first, assuming the language does not require a particular order.

Describe in your own words how the Carlos language allows handles recursive structs. Describe what kinds of restrictions the language definition imposes and why. Describe how the compiler enforces the restrictions. Write well. Use technical vocabulary accurately. An AI assistant can help you get your grammar and spelling right, though it is unlikely to get the right answer.

- CARLOS handles recursive structs by specifically not allowing them to be used. You are not allowed to have the type of a field be struct itself inside a struct as that would lead to an infinitely sized structure. As seen in the code snippet below, the analyzer goes through all the fields of the struct and makes sure that there is not another struct type in the declared struct, preventing the infinitely sized struct issue.



```
function includesAsField(structType, type) {
  // Whether the struct type has a field of type type, directly or indirectly
  return structType.fields.some(
    field =>
      field.type === type ||
      (field.type?.kind === "StructType" && includesAsField(field.type, type))
  )
}

function mustNotBeSelfContaining(structType, at) {
  const containsSelf = includesAsField(structType, structType)
  must(!containsSelf, "Struct type must not be self-containing", at)
}
```

Some languages do not have loops. Write a function, using tail recursion (and no loops) to compute the minimum value of an array or list in Python, C, JavaScript, and in either Go, Erlang, or Rust (your choice). Obviously these languages probably already have a min-value-in-array function in a standard library, but the purpose of this exercise is for you to demonstrate your understanding of tail recursion. Your solution must be in the classic functional programming style, that is, it must be stateless. Use parameters, not nonlocal variables, to accumulate values. Assume the array or list contains floating-point values.

```
def recursive_min(list, min_val=float('inf')):
    if len(list) == 0:
        return min_val
    return recursive_min(list[1:], min_val=min(list[0], min_val))
```

```
package main

import (
    "fmt"
    "math"
)

func minimum(arr []float64, minSoFar float64) float64 {
    if len(arr) == 0 {
        return minSoFar
    }

    return recursive_min(arr[1:], math.Min(arr[0], minValue))
}
```

Your friend creates a little JavaScript function to implement a count down, like so:

```
function countDownFrom10() {
    let i = 10;
    function update() {
        document.getElementById("t").innerHTML = i;
        if (i-- > 0) setTimeout(update, 1000);
    }
    update();
}
```

**Your other friend says "Yikes, you are updating a non-local variable! Here is a better way:"

```
function countDownFromTen() {
    function update(i) {
        document.getElementById("t").innerHTML = i;
            if (i-- > 0) setTimeout(update(i), 1000);
        }
    update(10);
}
```

What does your second friend's function do when called? Why does it fail? Your friend is on the right path though. Fix their code and explain why your fix works:

- This friend's function fails because the setTimeout function is not being used properly. This current implementation by friend two invokes the update function immediately instead of waiting until after the delay. To fix this, we need to wrap it inside an anonymous function:

```
function countDownFrom10() {
    function update() {
        document.getElementById("t").innerHTML = i;
        if (i-- > 0) setTimeout(() => update(i), 1000);
    }
    update(10);
}
```

Find as many linter errors as you can in this Java source code file (C.java):

- You can use SonarLint or FindBugs or FindSecBugs or PMD or whatever you prefer. You might even need to use a combination of tools because it is possible no tool finds them all. (Please note you are not expected to already know what all the issues are here. The idea is to practice with tools and have good discussions with teammates. Find as many as you can, and read and understand each problem that is reported to you so you learn (1) what kinds of potential bugs and security problems can exist even in compilable and runnable code, and (2) the kinds of things that a static analyzer *can* detect.)

```
import java.util.HashMap;
class C {
    static final HashMap<String, Integer> m = new HashMap<String, Integer>();

    static int zero() {
        return 0;
    }

    public C() {
    }
}
```

SonarLint:

- The file is not in a named package
- In the new HashMap declaration: Replace the type specification in this constructor call with the diamond operator ("<>"). (sonar.java.source not set. Assuming 7 or greater.)sonarqube(java:S2293)
- Public constructor should be hidden.
- Static int zero(): Remove this method and declare a constant for this value.sonarqube(java:S3400)
- public C() method: Add a nested comment explaining why this method is empty, throw an UnsupportedOperationException or complete the implementation.sonarqube(java:S1186)