Programming Language Concepts

Programming Language Theory

Topics

- Name and Binding
- **Environments and Blocks**
- Scope Rules

Referencing Environment

- An ***Environment*** is a set of bindings,
- for names and objects at a certain point of execution.
- Usually, we only refer to bindings not set up by language definition.

Blocks

- You may first think about `{ ... }`.
- A ***Block*** is a textual region of a program, identified a start and an end sign.
- A block can contain declarations local to that region.
    - i.e., such declarations are not valid outside that region.

Blocks

- Blocks can be represented in various ways.
- Usually, every time we enter and exit a block, the environment is changed.
- Blocks can be nested.
- Overlapping of blocks is never permitted.
  - i.e., we can't close a previous block, until the last opened block is closed.

Java

```java
if(a > 0 && b > 0) {
    q = a / b;
}
```

Python

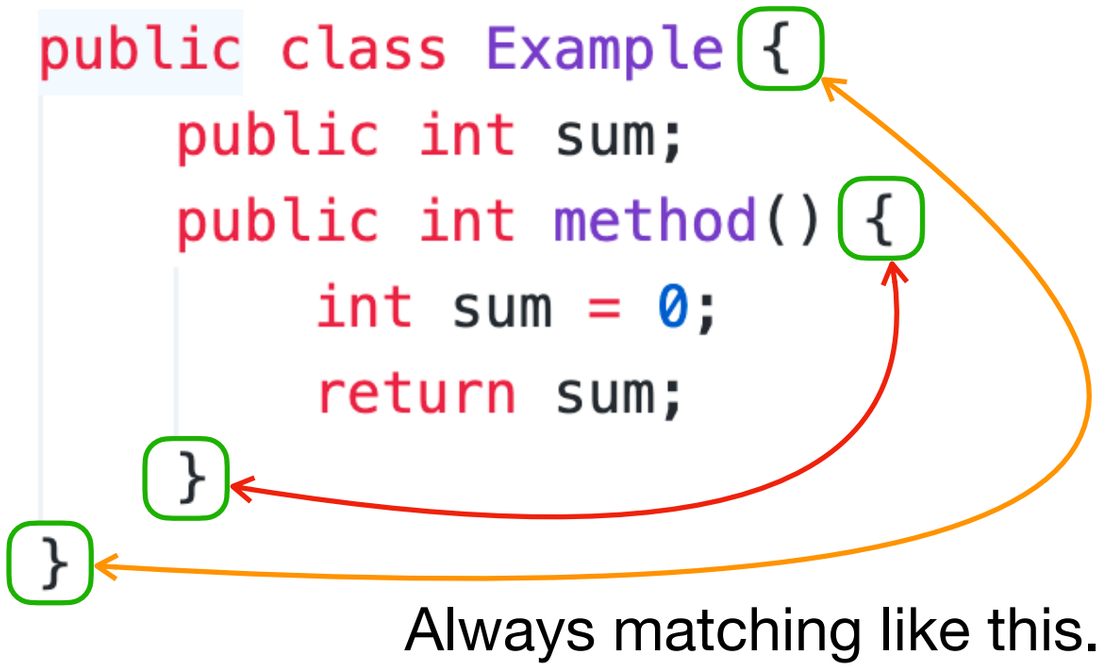
```python
if a > 0 and b > 0:
    q = a / b
```

Scheme

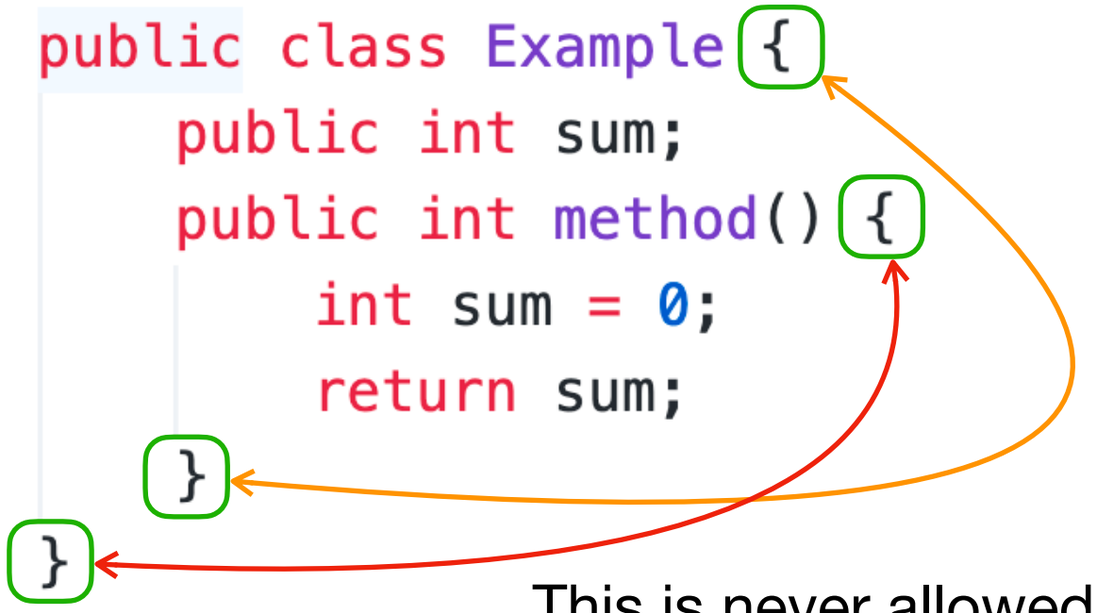
```scheme
(define var "PL")
(let ((var 10))

)
```

Overlapping of Blocks

- Usually, inside a block, we will consider a local environment.
- Overlapping will make it very complicated.
- Hence this is not permitted in any PL.
- However, block nesting policy can be slightly different PLs.



Always matching like this.



This is never allowed.

Types of Environment

- The environment associated with a block can be composed of the followings.

- ***Local environment*** is a set of bindings for names declared locally in the block.

- ***Non-local environment*** is formed of bindings for names visible, but not declared in the block.

- ***Global environment*** is the environment from bindings created when the program begins.

Visibility Rules

- This is somewhat informal concept.

- A ***local declaration*** in a block is ***visible*** to the block, and all the other blocks inside that block.

- If there is a new declaration of the same name in a block, this new declaration ***hides*** the previous one.

Visibility Rules

- There are block 0~3, and their ranges are represented by thin grey lines.

- When the current block is changed, the environment is changed.

- Hence the same name can be linked to a different object.

- What are the values of variables c and d?

```
0:{int a = 1;
    1:{int b = 2;
        2:{    int b = 3;
               int c = a + b;
               printf("%d\n", c);
           }
        3:{    int d = a + b;
               printf("%d\n", d);
           }
    }
}
```

Visibility Rules

- First of all, a is declared in block 0, hence it is visible to all blocks 0~3.
- The first b is declared in block 1, hence it is visible to blocks 1, 2, 3.
- The second b is declared in block 2, and visible to block 2 only.
- It also hides the first b in block 2, hence in block 2, b always denotes the second one.

```
0:{int a = 1;
    1:{int b = 2;
        2:{    int b = 3;
               int c = a + b;
               printf("%d\n", c);
           }
        3:{    int d = a + b;
               printf("%d\n", d);
           }
    }
}
```

Visibility Rules

- On the other hand, in block 3, the second b (in block 2) is not visible.

- Still, the first b in block 1 is visible to block 3, hence it is used to compute d.

- Therefore c is 4, and d is 3.

```
0: {int a = 1;
    1: {int b = 2;
        2: {    int b = 3;
                int c = a + b;
                printf("%d\n", c);
            }
        3: {    int d = a + b;
                printf("%d\n", d);
            }
    }
}
```

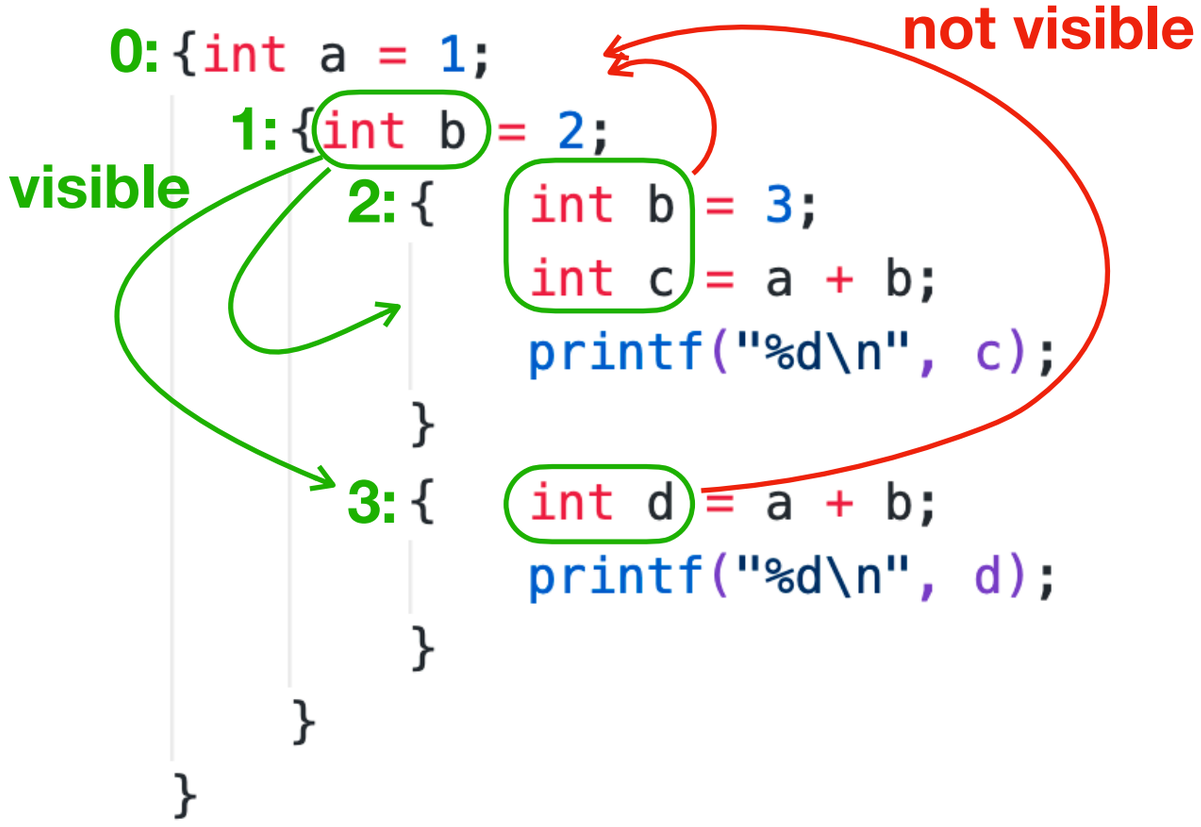
Environments

- Let's suppose that variable *a* is a global variable.
- Then *a* is visible to all blocks, and a part of global environment.
- For block 1, the binding of *a* is ***global*** as well as ***non-local*** environment.

```
0: {int a = 1;
    1: {int b = 2;
        2: {    int b = 3;
                int c = a + b;
                printf("%d\n", c);
           }
        3: {    int d = a + b;
                printf("%d\n", d);
           }
    }
}
```

Environments

- Names in local environment is ***visible to inner blocks***.
- While names in local environment is ***not visible to outer blocks***.
- Names in non-local environment are ***hidden by the same name*** in local environment.
- More precisely, the binding of the first b *is deactivated* in local environment of block 2.



Still, This is not enough

- Visibility rules we discussed are roughly describing the big picture.

- Specific and detailed rules could be different in different PLs.

- For example, the case we just described is not valid in Java.

Java Example

- The previous example written in C, and works without errors.
- In Java, duplicate local variable is not allowed.
- On the other hand, we can still override a global variable.
- Therefore we have to understand specific rules for each programming language.

```
{int a = 1;
    {int b = 2;
        {   int b = 3;
            int c = a + b;
            System.out.println(c);
        }
        {   int d = a + b;
            System.out.println(d);
        }
    }
}
```

```
int b - Example1.main(String[])
Duplicate local variable b Java(536870967)
```

Summary

- Blocks
- Association between blocks and environments.
- Visibility Rules