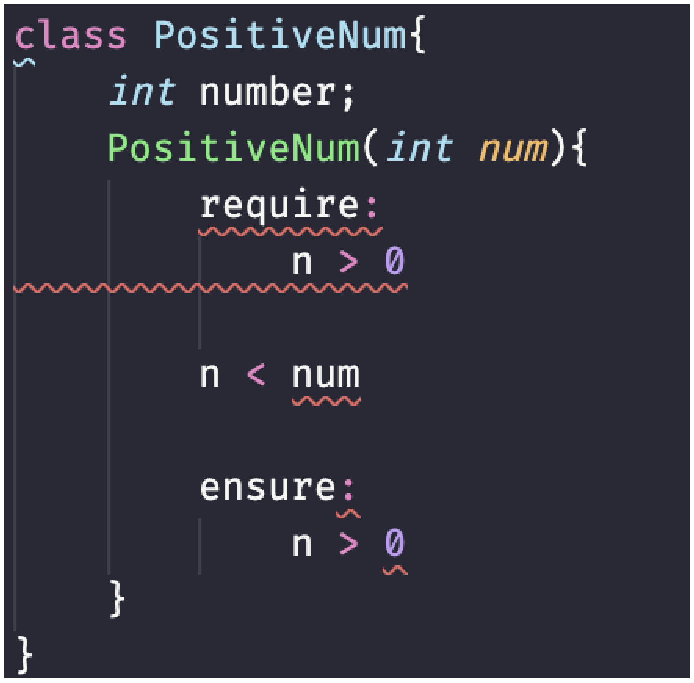
Names:

Lucas Gama : 108696592

Felipe Lima : 109290055

1. In Object Oriented Programming, Design by Contract is when the client and the server are bound by a contract. In other words, the server promises to do its job (formalized by the postconditions) as long as the client accomplishes a minimum requirement. The server can only do the job if the client uses the server correctly, as defined in the pre-conditions. For example, in an application that has the goal of creating a file, files will be created by the server as long as the client fulfills its pre-conditions. If a client fails to complete the pre-conditions correctly, anything can happen as the server is no longer obligated to complete its promises. Basically, Contract by Design states how a client should use a server and how a server can implement the interface. The difference between explicit and implicit contracts is that explicit contracts will have specifications for a routine explicitly written in code. Differently, implicit contracts won't have specifications written in code.
    a. Below, we can see a pseudo code example of a Design by Contract



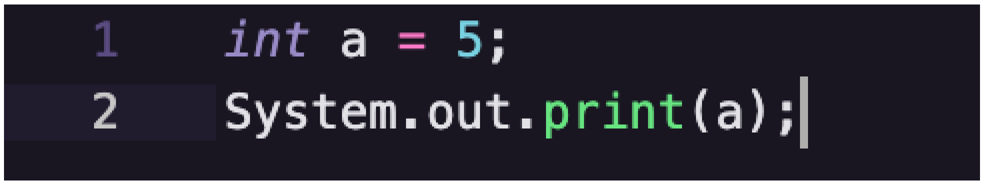
```
class PositiveNum{
    int number;
    PositiveNum(int num){
        require:
            n > 0

        n < num

        ensure:
            n > 0
    }
}
```

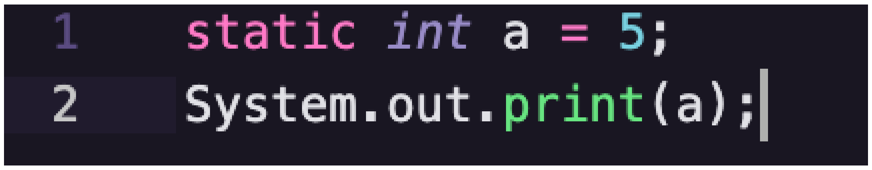
Source: link_1, link_2

2. Java interfaces differ from the pure OO interface intent in some different ways. The first is that the idea of object oriented programming is that it should only have objects, and java has 8 primitive data types: char, boolean, byte, short, int, long, float, double which are not objects and can be used without the use of any objects.



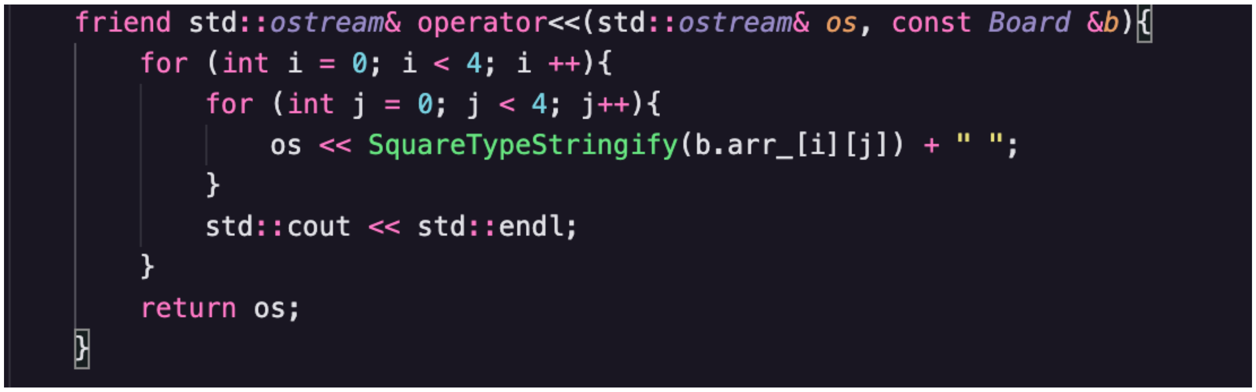
```java
int a = 5;
System.out.print(a);
```

The second is the static keyword . In pure object oriented language, everything should be accessed through objects, and with the static keyword can be used without the use of an object .



```java
static int a = 5;
System.out.print(a);
```

The third is that java does not support some object oriented concepts like multiple inheritance and operator overloading.

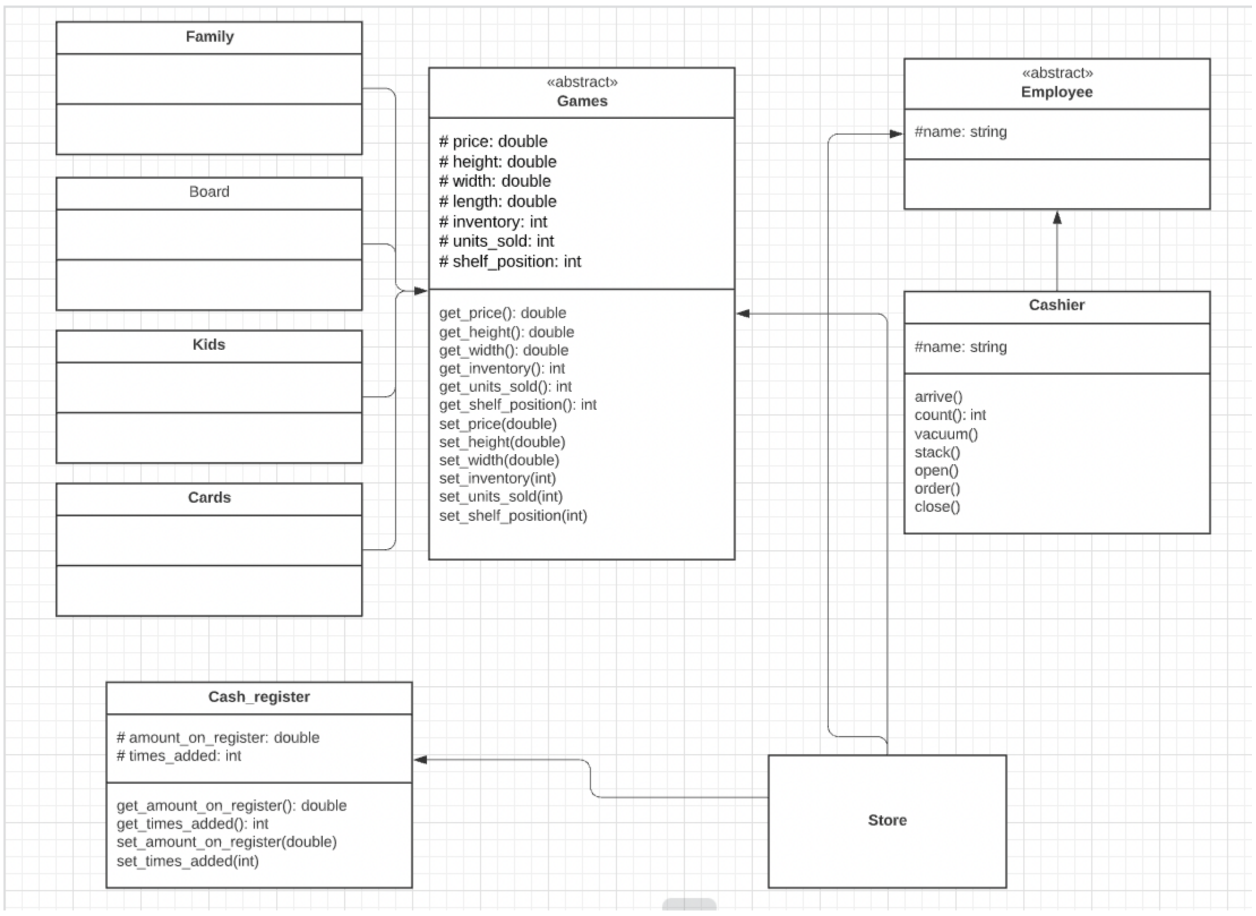


```cpp
friend std::ostream& operator<<(std::ostream& os, const Board &b){
    for (int i = 0; i < 4; i ++){
        for (int j = 0; j < 4; j++){
            os << SquareTypeStringify(b.arr_[i][j]) + " ";
        }
        std::cout << std::endl;
    }
    return os;
}
```

(code is in c++ do demonstrate operator overloading)

Sourcers: Link1, Link2, link3

3.


Family
Board
Kids
Cards
«abstract»
Games
# price: double
# height: double
# width: double
# length: double
# inventory: int
# units_sold: int
# shelf_position: int
get_price(): double
get_height(): double
get_width(): double
get_inventory(): int
get_units_sold(): int
get_shelf_position(): int
set_price(double)
set_height(double)
set_width(double)
set_inventory(int)
set_units_sold(int)
set_shelf_position(int)
«abstract»
Employee
#name: string
Cashier
#name: string
arrive()
count(): int
vacuum()
stack()
open()
order()
close()
Cash_register
# amount_on_register: double
# times_added: int
get_amount_on_register(): double
get_times_added(): int
set_amount_on_register(double)
set_times_added(int)
Store