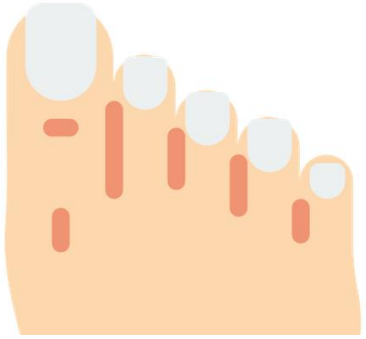
OOP

1. Typed languages
2. Object/Class
3. Aggregation
4. Encapsulation
5. Inheritance
6. Inheritance
7. Polymorphism

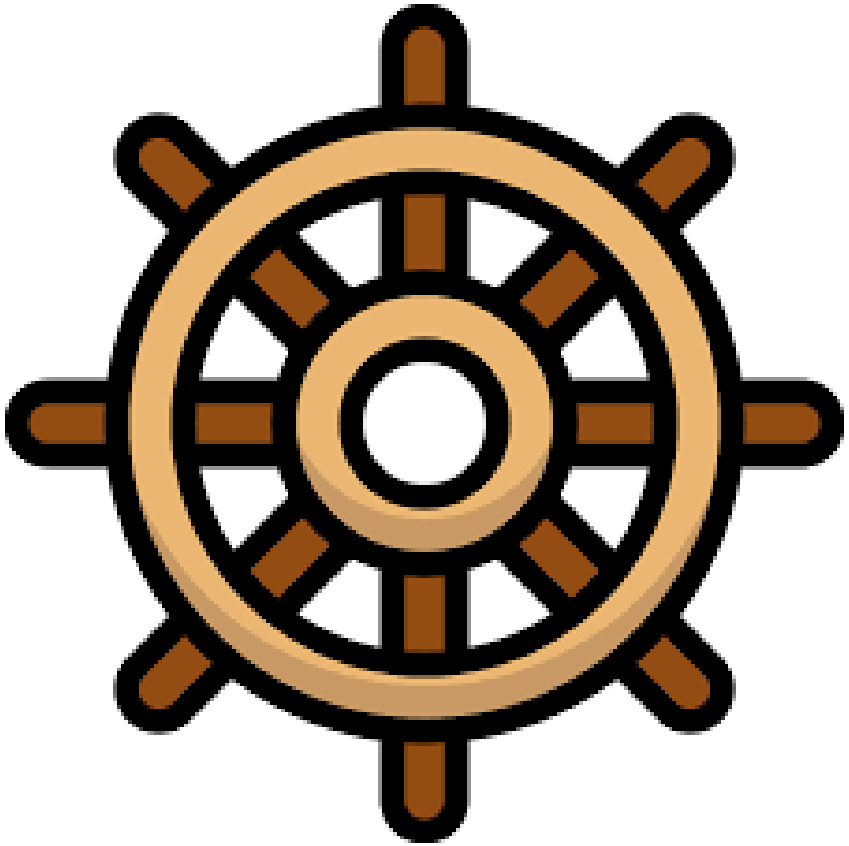
UML multiplicities

1 Exactly 1 instance	* 0 to many
0..1 0 or 1 instance	+ 1 to many
5 Exactly 5 instances	

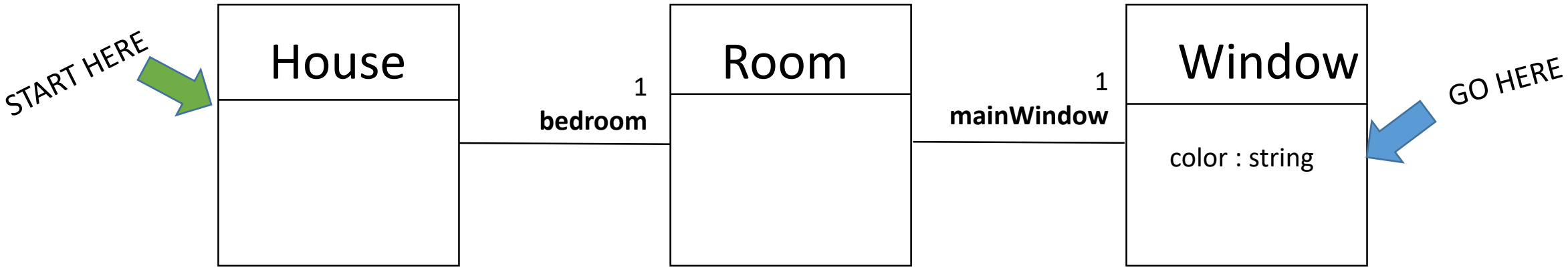
*Can you assign the **multiplicity***
Of those elements related to a man ?



NAVIGATION

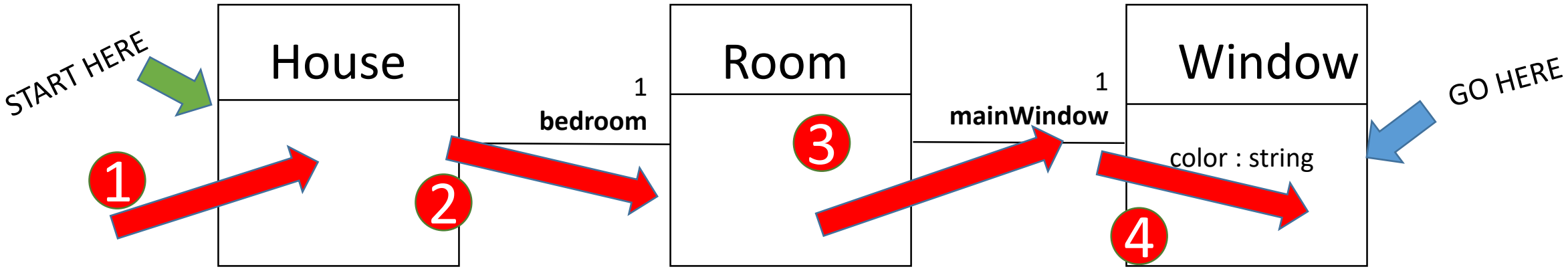


Complete this code to get the color of the bedroom main window



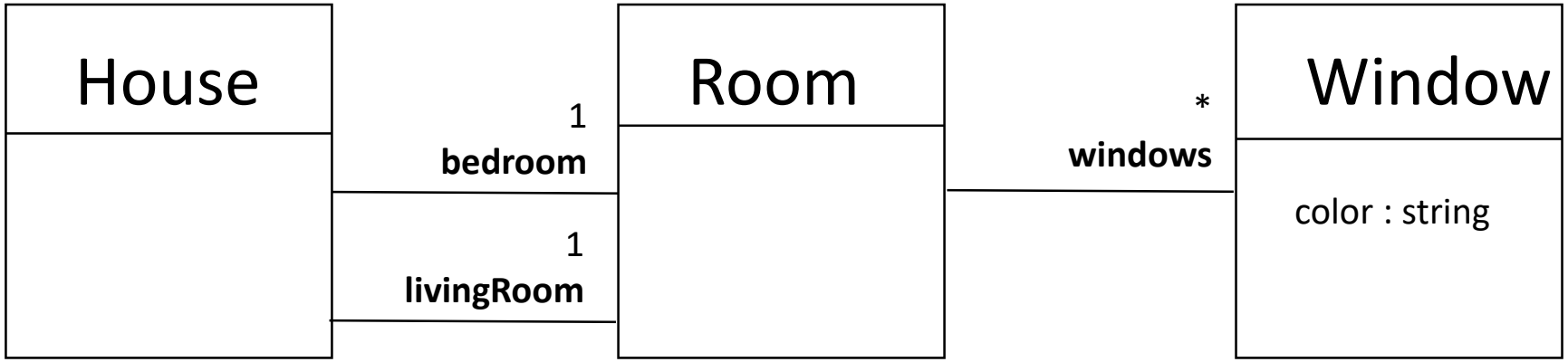

```
let sophanaHouse: House;  // Note : We don’t detail the initialization

Let theColor= sophanaHouse. ??
```

Complete this code to get the color of the bedroom main window



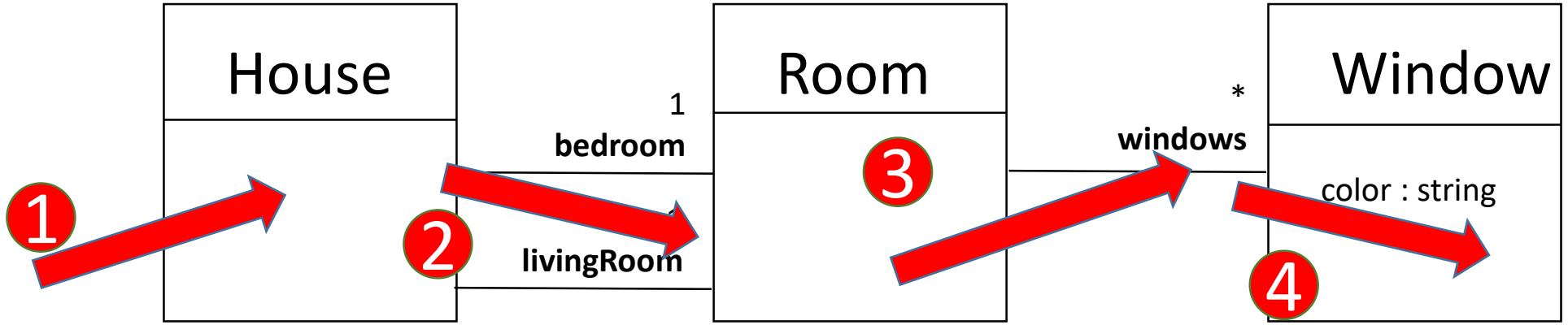

```
let sophanaHouse: House;  // Note : We don't detail the initialization

Let theColor= sophanaHouse.bedroom.mainWindow.color
```

Complete this code to get the color of the **first** window of the living room



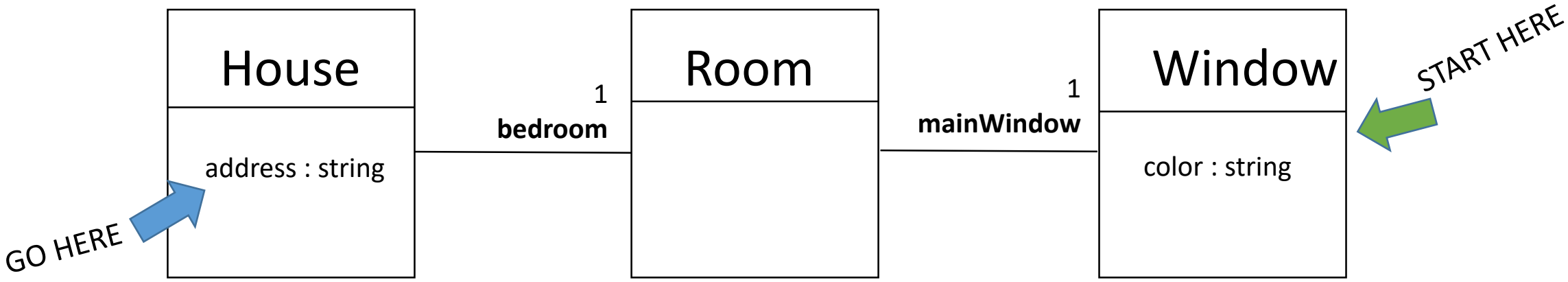

```
let sophanaHouse: House;  // Note : We don’t detail the initialization

Let theColor= sophanaHouse. ??
```

Complete this code to get the color of the **first** window of the living room




```
let sophanaHouse: House;  // Note : We don’t detail the initialization

Let theColor= sophanaHouse.livingRoom.windows[0].color;
```

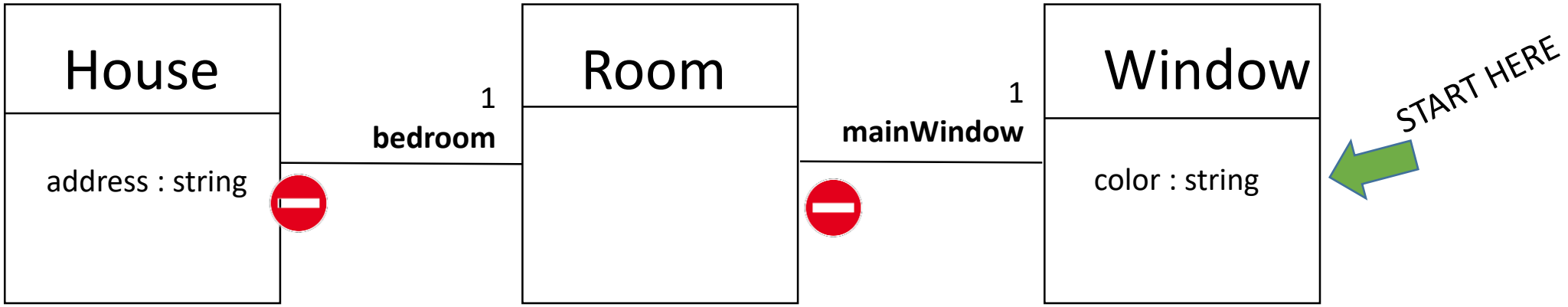
Can you access to the **house address** from the **bedRoomWindow** object ?




```
let bedRoomWindow: Window;  // Note : We don't detail the initialization

Let houseAddress= bedRoomWindow. ??
```

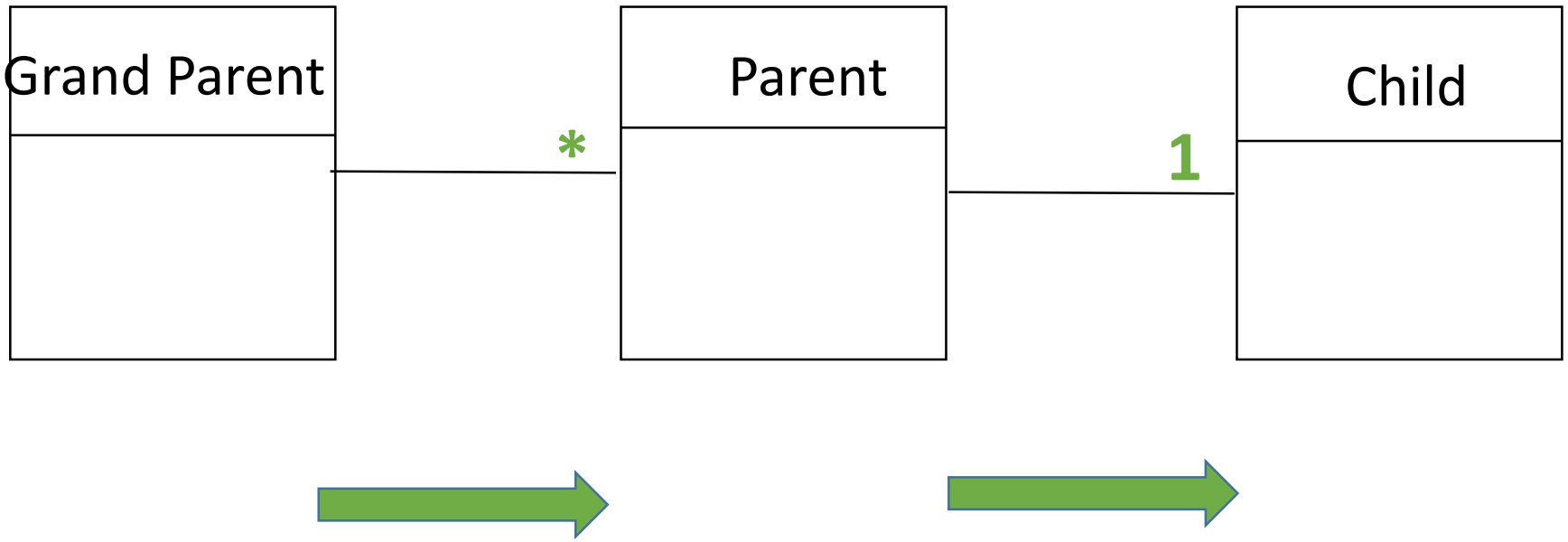
Can you access to the **house address** from the **bedRoomWindow** object ?

- The windows does not have a reference to the parent room
- The room does not have a reference to the house



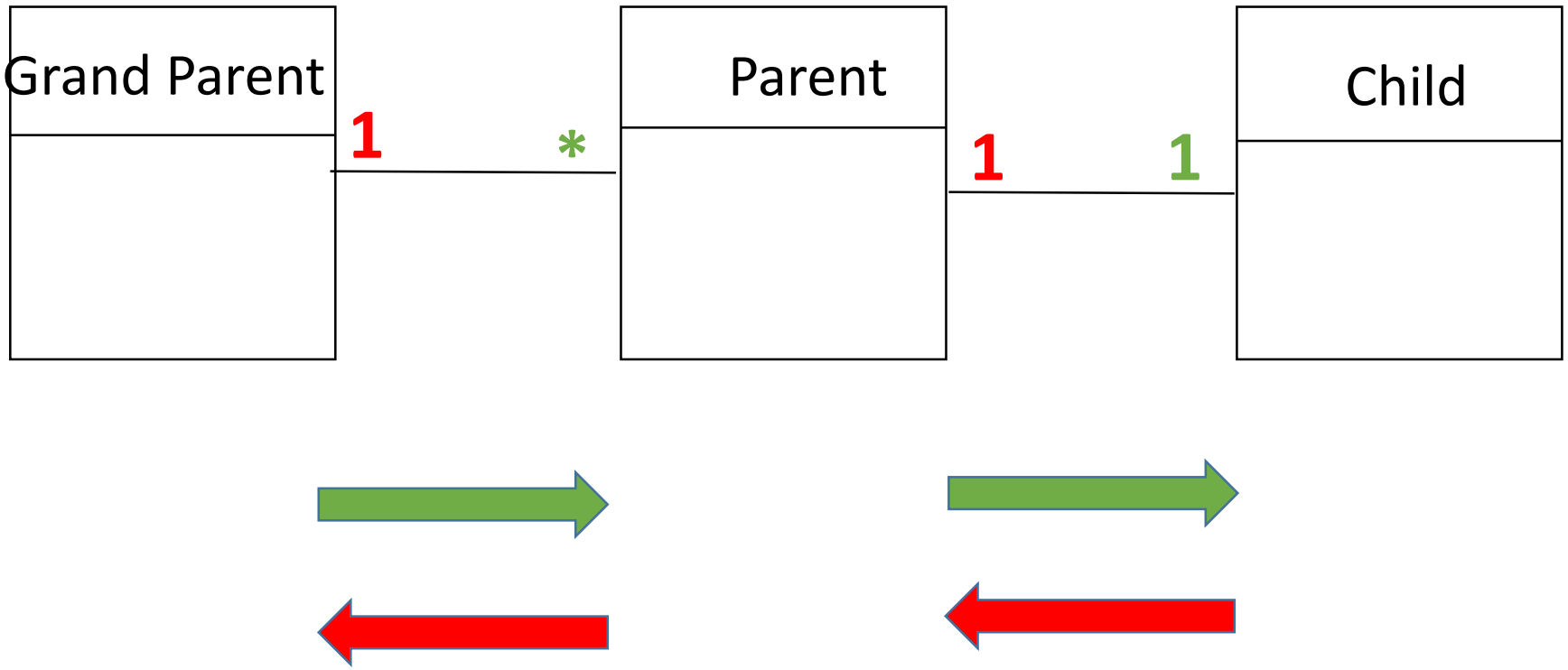
Mono directional association

Browse model from parent to children



Bi directional association

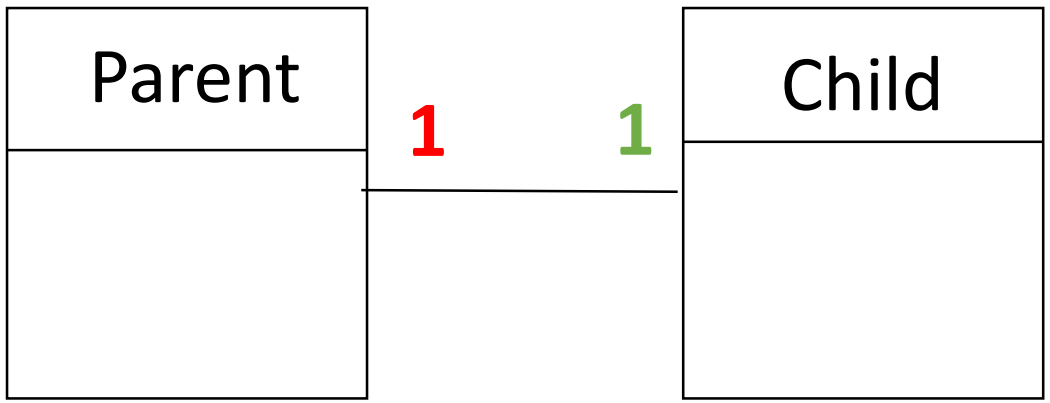
Browse model from children to parent



The parent knows the child

+

The child knows the parent



```
class Parent {
  child?: Child;
}

class Child {
  parent?: Parent;
}

// Create the objects
let the_parent = new Parent();
let the_child = new Child();

// make the links
the_parent.child = the_child;
the_child.parent = the_parent;
```

ENCAPSULATION

Is this code is safe ? Why ?

```
class BankAccount {
  name: string;
  balance: number = 0;

  constructor(name: string) {
    this.name = name;
  }
}

let ronanAccount = new BankAccount("ronan");
ronanAccount.balance -= 1000; // Debit 1000 dollars  !!!!
```

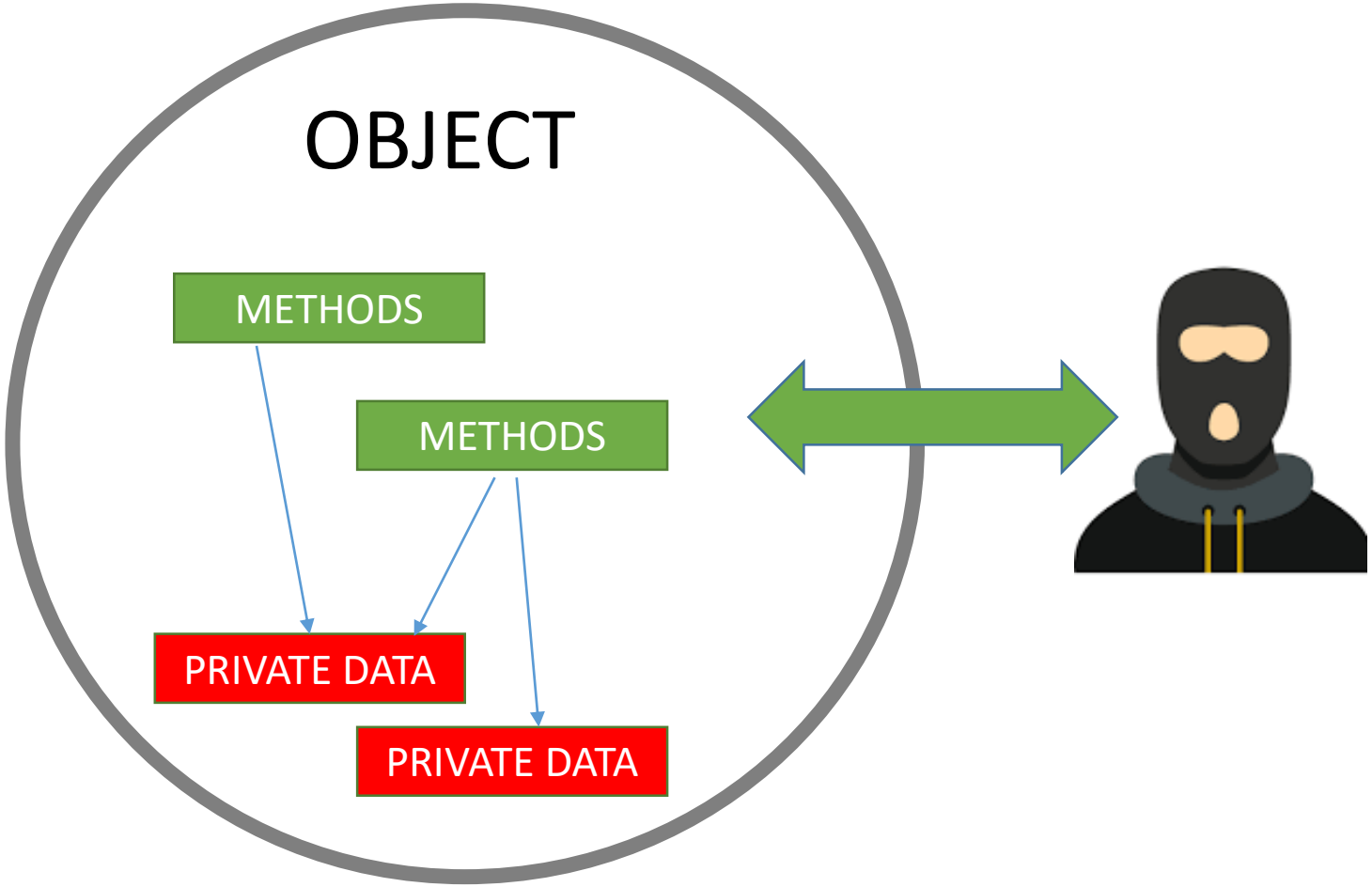
Let's put our data private

Let's add a method to debit money

Now the Operation Will not be Possible !!

```
class BankAccount {
  private name: string;
  private balance: number = 0;

  constructor(name: string) {
    this.name = name;
  }

  debit(amount:number) {
     if (this.balance - amount >=0 ) {
         this.balance -= amount;
     }
  }



}

let ronanAccount = new BankAccount("ronan");
ronanAccount.debit(1000);
```

In OOP we want to protect our data !!!



A getter is to get data

```
class BankAccount {
  private name: string;

  getName() : string {
    return this.name;
  }
```

A setter is to set data

```
class BankAccount {
  private name: string;

  setName(newName:string) {
    this.name = newName;
  }
```

What this will code display ?

```
class Bob {
  private name: string = "test";

  getName() {
    return this.name;
  }
}

let myBob = new Bob();
let bobName = myBob.getName();
bobName = "ronan";

console.log(myBob.getName());
```

A - ronan

B - test

C – error

ACTIVITY 1

```
class Rectangle {
  width: number;
  height: number;
  area: number;

  constructor(width: number, height: number) {
    this.width = width;
    this.height = height;
    this.area = this.width * this.height;
  }
}

let smallRectangle = new Rectangle(4, 8);
smallRectangle.width = 50;
```

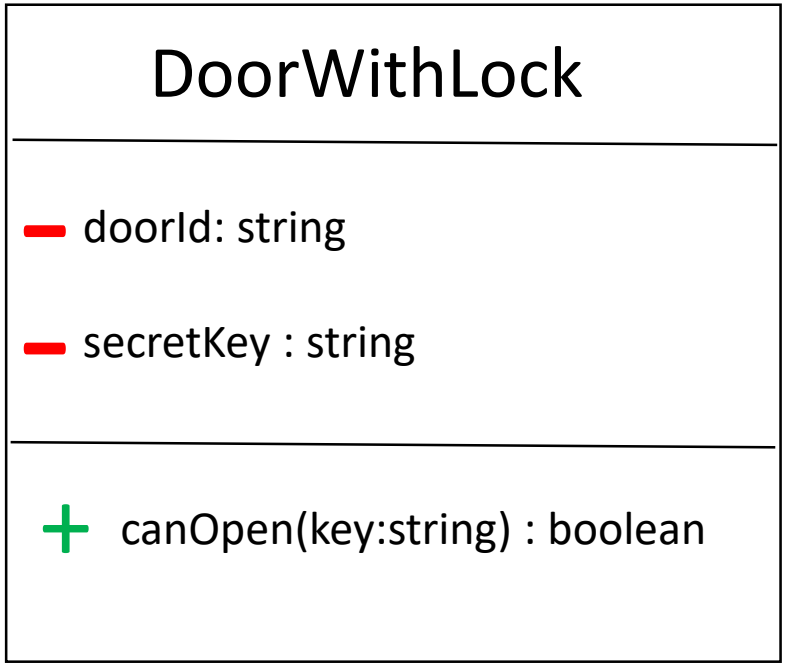
// 1-Look at the code : what is the problem **after you changed the width to 50** ?

// 2 –Fix this code by **encapsulating the data and adding methods**

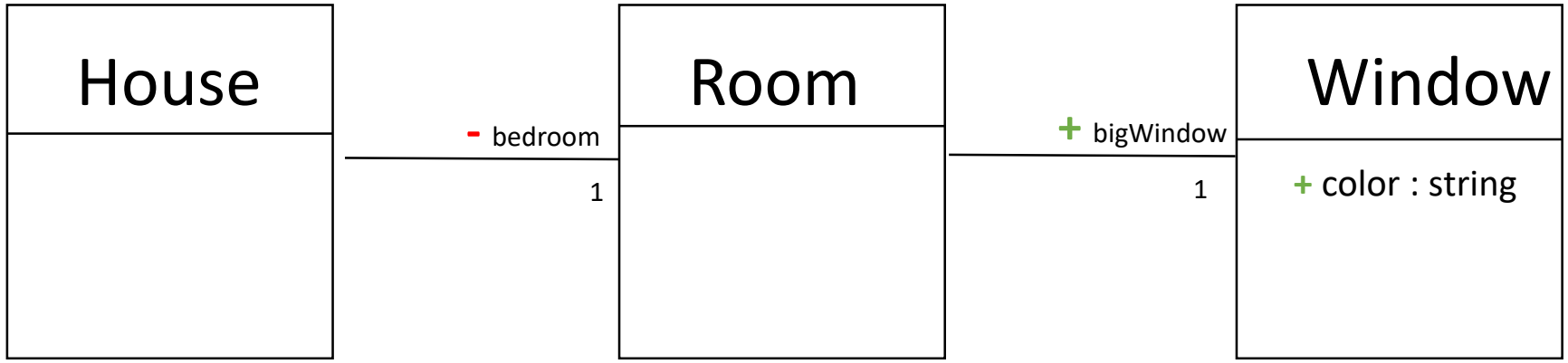
Let’s update our UML class diagram !

− PRIVATE + PUBLIC

Only the Method is Public here



DOES THIS CODE WILL RAISE AN ERROR ?

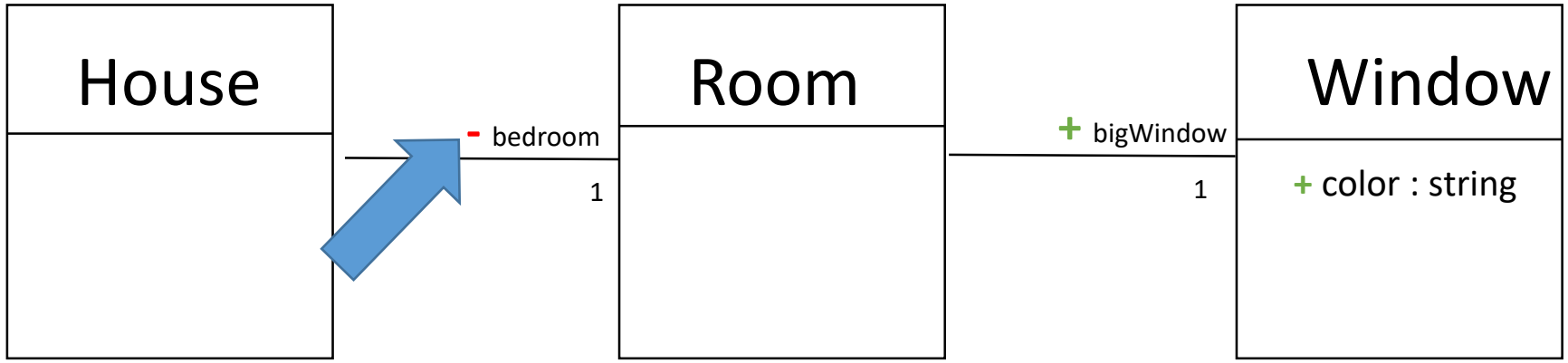



```
let sophanaHouse = new House();
Let theColor= sophanaHouse.bedroom.bigWindow.color
```

A No

B YES

DOES THIS CODE WILL RAISE AN ERROR ?

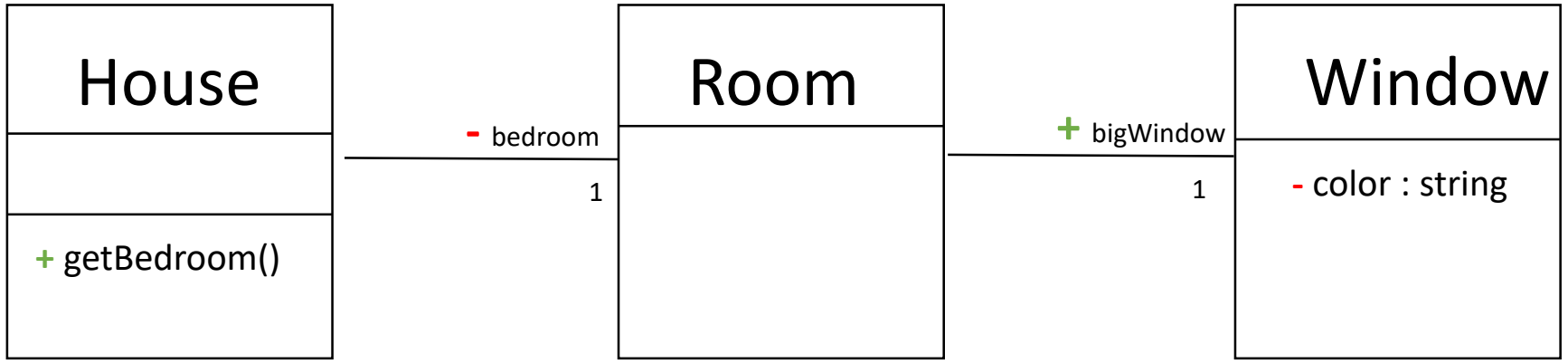



```
let sophanaHouse = new House();
Let theColor= sophanaHouse.bedroom.bigWindow.color
```

A YES

B YES

DOES THIS CODE WILL RAISE AN ERROR ?

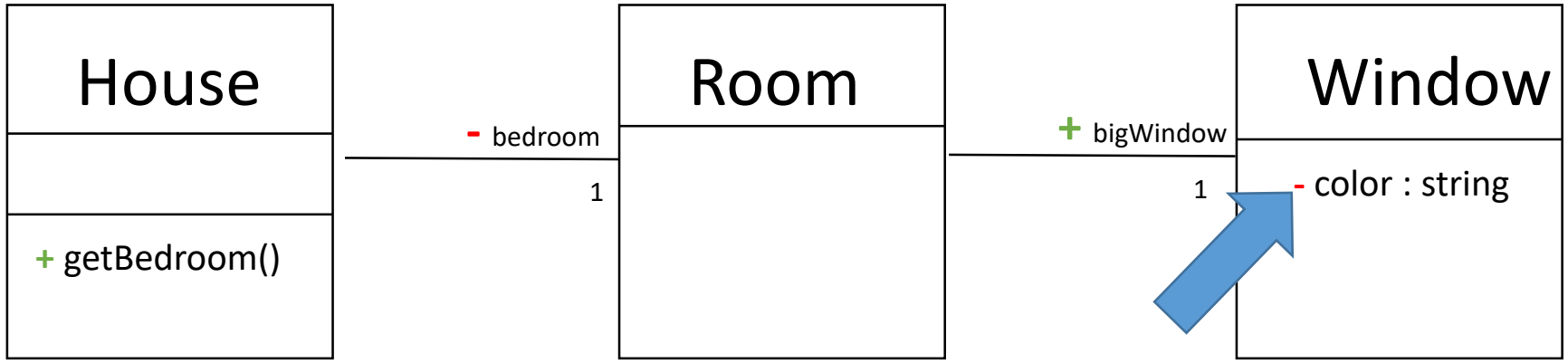



```
let sophanaHouse = new House();
Let theColor= sophanaHouse.getBedroom().bigWindow.color
```

A YES

B YES

DOES THIS CODE WILL RAISE AN ERROR ?




```
let sophanaHouse = new House();
Let theColor= sophanaHouse.getBedroom().bigWindow.color
```

A YES

B YES

TO SUM UP

- ✓ UML **multiplicities** : *1, 0..1, 5 , many etc.*

- ✓ We sometimes need to **navigate in 2 directions** in our model
  - ✓ This requires to keep the reference of the “parent” in the “child”

- ✓ We need to protect our data, and provide methods to set or get the data

WANT TO GO FURTHER ?

CLASSES & OBJECTS IN TYPESCRIPT

https://www.typescriptlang.org/docs/handbook/classes.html