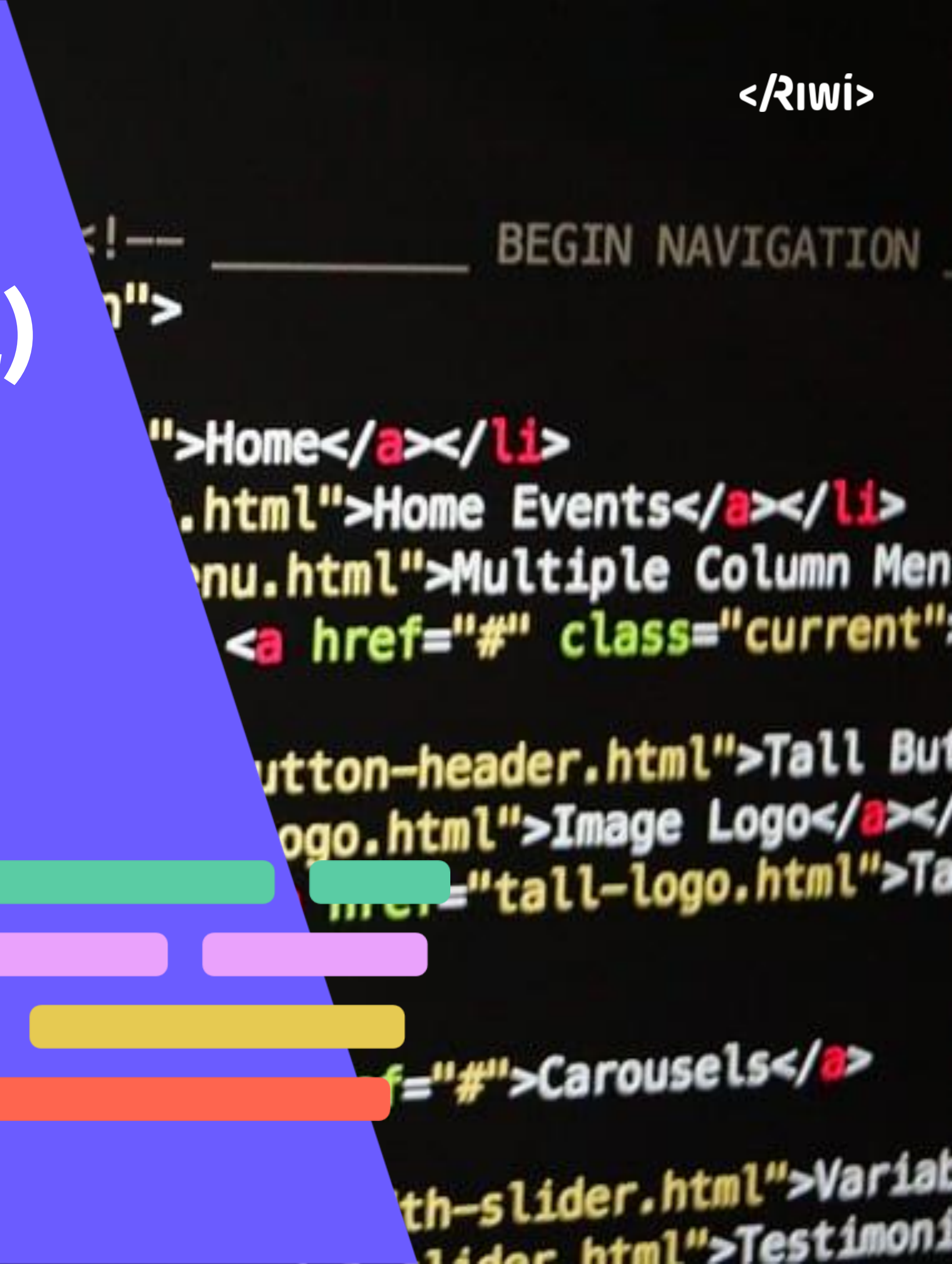
</Riwi>

Somos un ecosistema de desarrolladores de software

</Riwi>

Java Script POO (Document Object Model)



```
<!-- __________ BEGIN NAVIGATION
n">

">Home</a></li>
.html">Home Events</a></li>
nu.html">Multiple Column Men
<a href="#" class="current"
utton-header.html">Tall But
ogo.html">Image Logo</a></
="tall-logo.html">Ta
f="#">Carousels</a>
th-slider.html">Variab
```

POO

Object-oriented programming (OOP)

➢ Object-oriented programming (OOP) is a programming paradigm based on the concept of objects;

➢ We use objects to model (describe) real-world or abstract features;

➢ Objects may contain data (properties) and code (methods). By using objects, we pack data and the corresponding behavior into one block;

➢ In OOP, objects are self-contained pieces/blocks of code;

➢ Objects are building blocks of applications, and interact with one another;

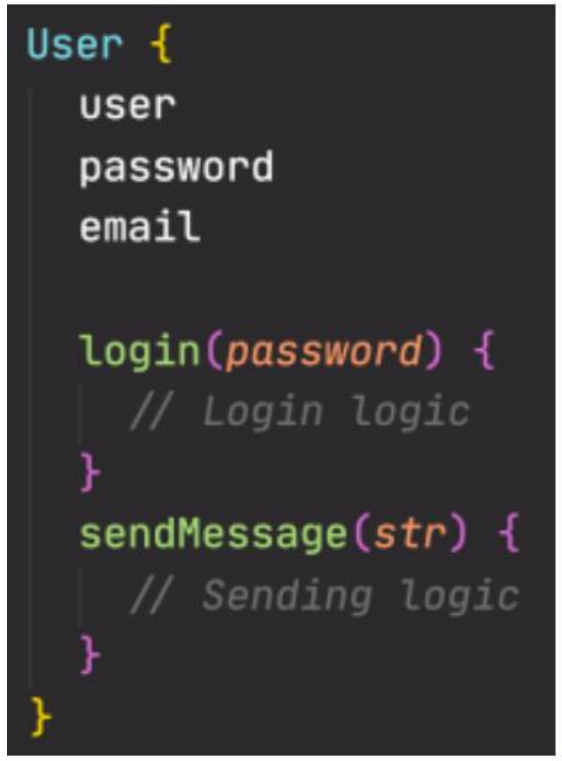
➢ Interactions happen through a public interface (API): methods that the code outside of the object can access and use to communicate with the object;

➢ OOP was developed with the goal of organizing code, to make it more flexible and easier to maintain (avoid “spaghetti code”).

POO

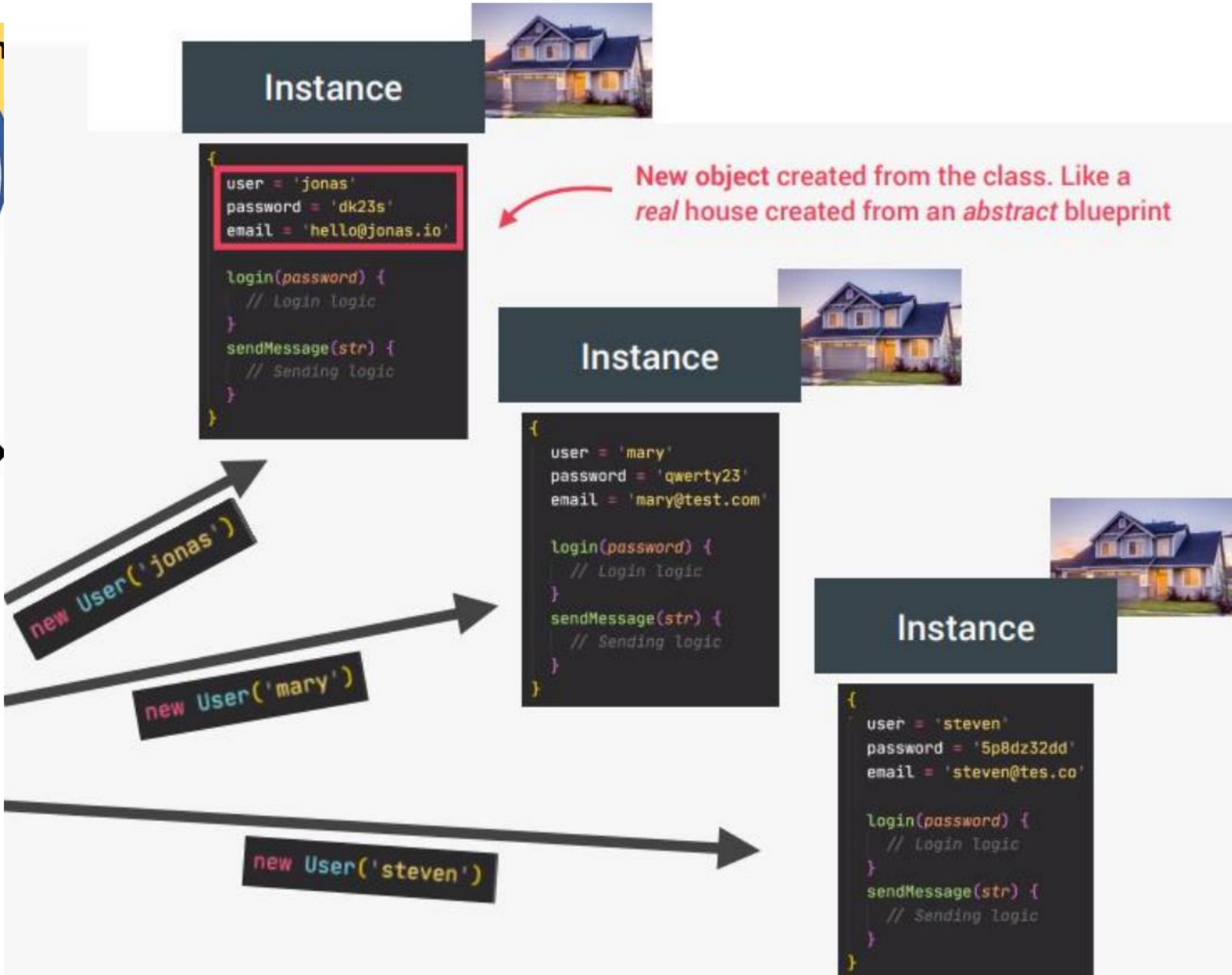
Classes and instances (Traditional OPP)

Like a blueprint from which we can create new objects



```
User {
    user
    password
    email

    login(password) {
        // Login logic
    }
    sendMessage(str) {
        // Sending logic
    }
}
```

Just a representation, NOT actual JavaScript syntax! JavaScript does NOT support real classes like represented here



New object created from the class. Like a *real* house created from an *abstract* blueprint

```
new User('jonas')
```

```
{
  user = 'jonas'
  password = 'dk23s'
  email = 'hello@jonas.io'

  login(password) {
    // Login logic
  }
  sendMessage(str) {
    // Sending logic
  }
}
```

```
new User('mary')
```

```
{
  user = 'mary'
  password = 'qwerty23'
  email = 'mary@test.com'

  login(password) {
    // Login logic
  }
  sendMessage(str) {
    // Sending logic
  }
}
```

```
new User('steven')
```

```
{
  user = 'steven'
  password = '5p8dz32dd'
  email = 'steven@tes.co'

  login(password) {
    // Login logic
  }
  sendMessage(str) {
    // Sending logic
  }
}
```

POO

The 4 fundamental OOP principles

ABSTRACTION

ENCAPSULATION

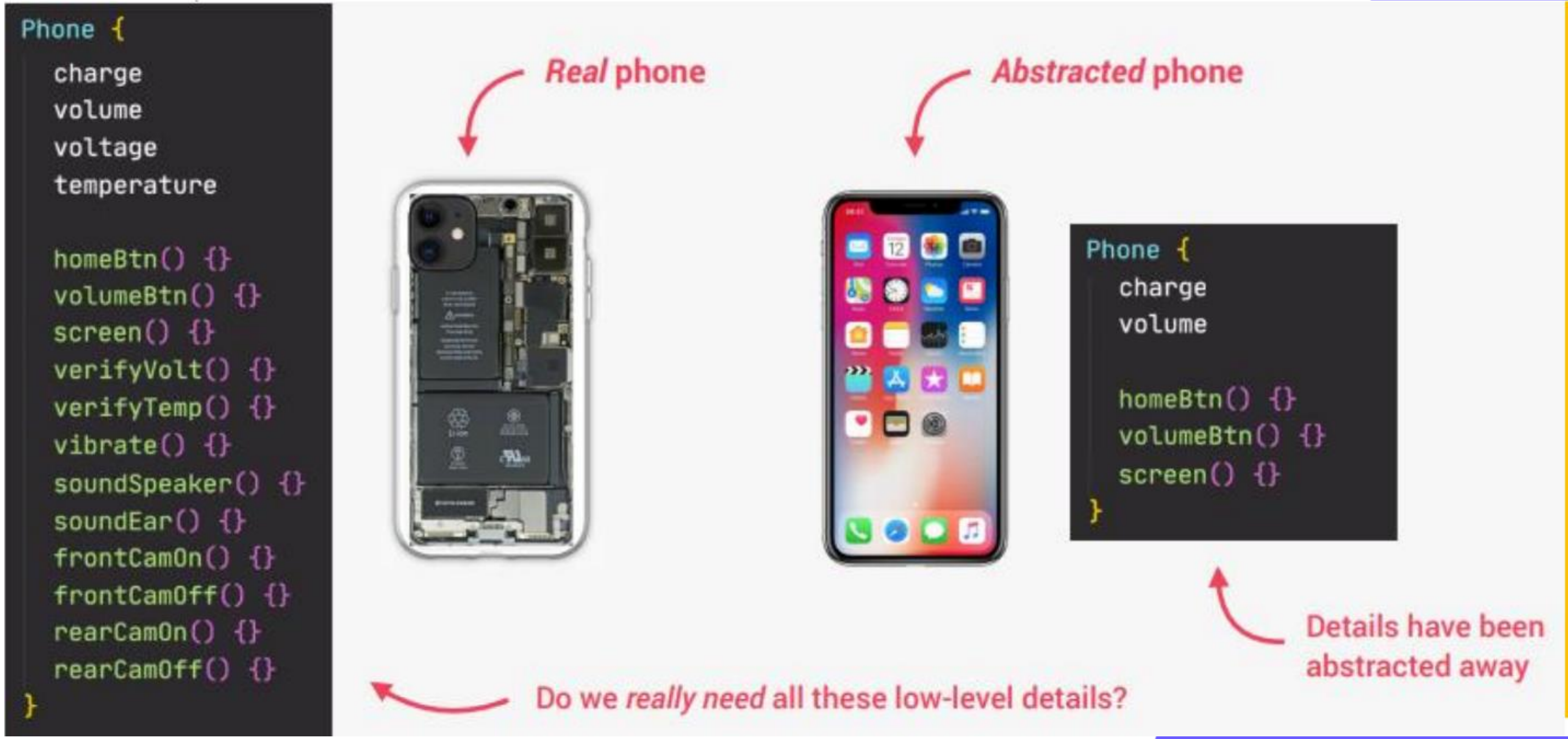
INHERITANCE

POLYMORPHISM

"How do we actually design classes? How do we model real-world data into classes?"

POO

ABSTRACTION



```
Phone {
    charge
    volume
    voltage
    temperature

    homeBtn() {}
    volumeBtn() {}
    screen() {}
    verifyVolt() {}
    verifyTemp() {}
    vibrate() {}
    soundSpeaker() {}
    soundEar() {}
    frontCamOn() {}
    frontCamOff() {}
    rearCamOn() {}
    rearCamOff() {}
}
```

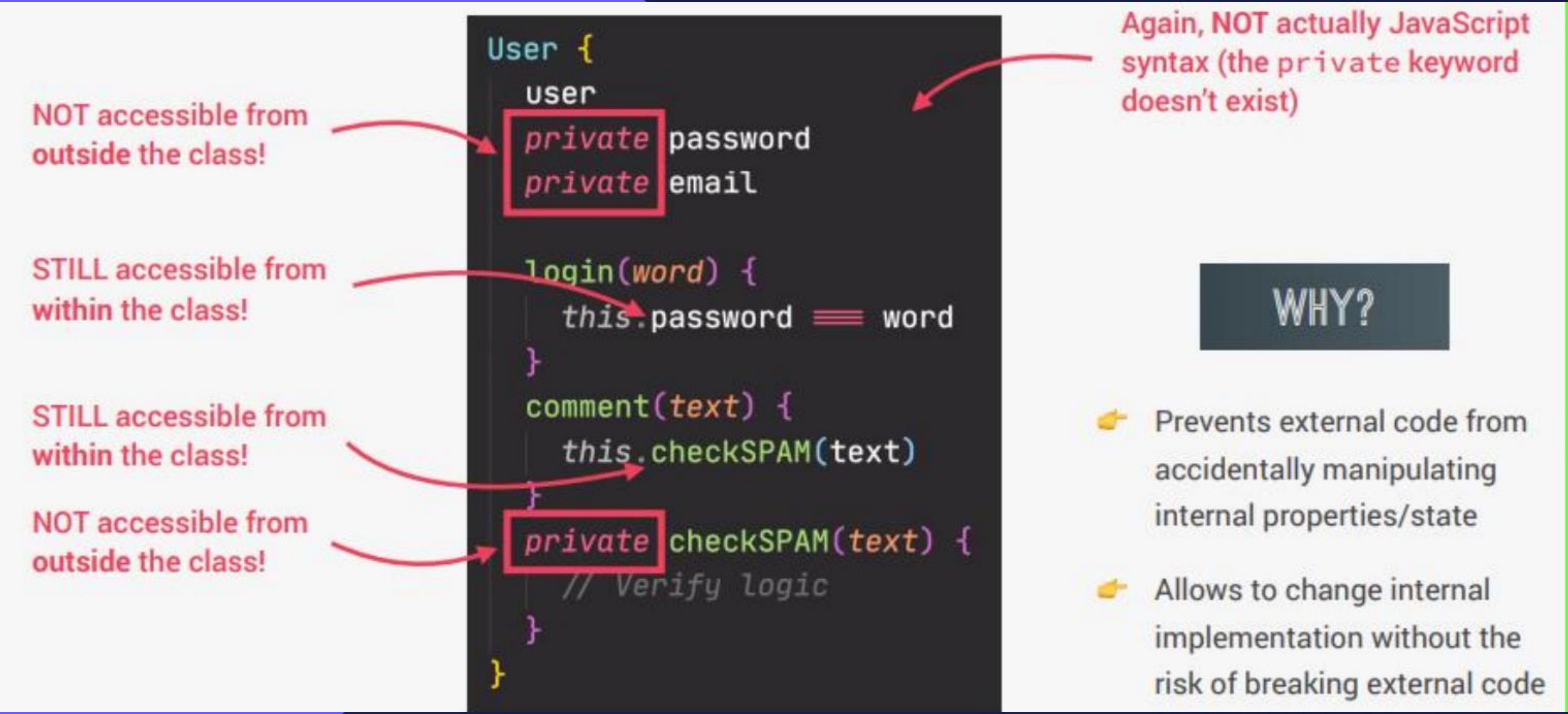
```
Phone {
    charge
    volume

    homeBtn() {}
    volumeBtn() {}
    screen() {}
}
```

Real phone
Abstracted phone
Details have been abstracted away
Do we *really need* all these low-level details?

Abstraction: Ignoring or hiding details that don't matter, allowing us to get an overview perspective of the thing we're implementing, instead of messing with details that don't really matter to our implementation.

POO

ENCAPSULATION



NOT accessible from outside the class!

STILL accessible from within the class!

STILL accessible from within the class!

NOT accessible from outside the class!

```
User {
  user
  private password
  private email

  login(word) {
    this.password === word
  }
  comment(text) {
    this.checkSPAM(text)
  }
  private checkSPAM(text) {
    // Verify logic
  }
}
```

Again, NOT actually JavaScript syntax (the `private` keyword doesn't exist)

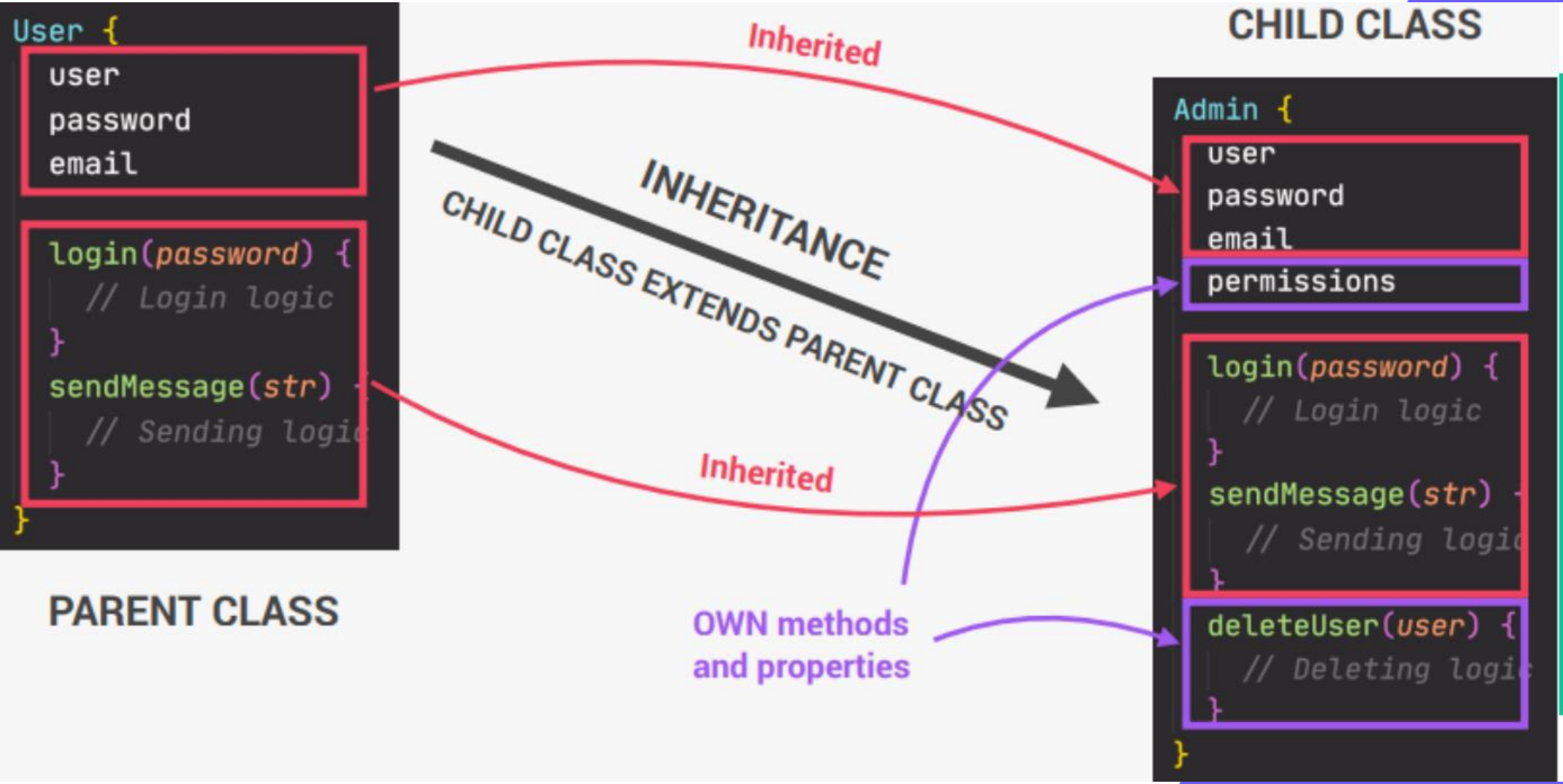
WHY?

- 👉 Prevents external code from accidentally manipulating internal properties/state
- 👉 Allows to change internal implementation without the risk of breaking external code

Encapsulation Keeping properties and methods private inside the class, so they are not accessible from outside the class. Some methods can be exposed as a public interface (API).

POO

INHERITANCE



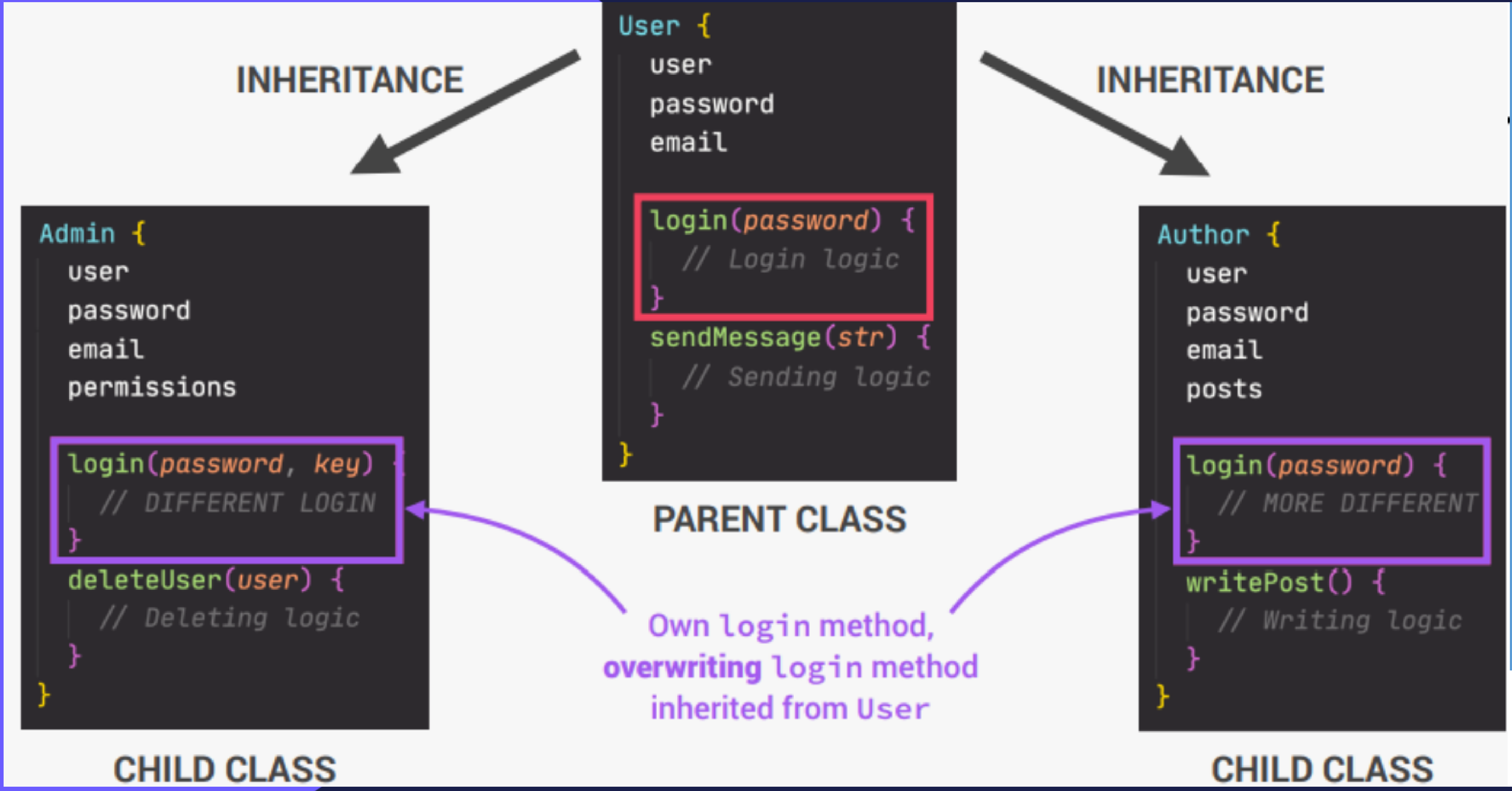
```
User {
  user
  password
  email

  login(password) {
    // Login logic
  }
  sendMessage(str) {
    // Sending logic
  }
}
```

```
Admin {
  user
  password
  email
  permissions

  login(password) {
    // Login logic
  }
  sendMessage(str) {
    // Sending logic
  }
  deleteUser(user) {
    // Deleting logic
  }
}
```

Inheritance: Making all properties and methods of a certain class available to a child class, forming a hierarchical relationship between classes. This allows us to reuse common logic and to model real-world relationships.

POO

POLYMORPHISM


INHERITANCE
INHERITANCE
PARENT CLASS
CHILD CLASS
CHILD CLASS
Own login method, overwriting login method inherited from User

```
User {
  user
  password
  email

  login(password) {
    // Login logic
  }
  sendMessage(str) {
    // Sending logic
  }
}
```

```
Admin {
  user
  password
  email
  permissions

  login(password, key) {
    // DIFFERENT LOGIN
  }
  deleteUser(user) {
    // Deleting logic
  }
}
```

```
Author {
  user
  password
  email
  posts

  login(password) {
    // MORE DIFFERENT
  }
  writePost() {
    // Writing logic
  }
}
```

Polymorphism
A child class can overwrite a method it inherited from a
parent class [it's more complex that that, but enough for our purposes]

</Be a coder>