sui

Making Your First Smart Contract In **Sui Move**

Twitter: @0xShayan

February 15, 2023

Agenda

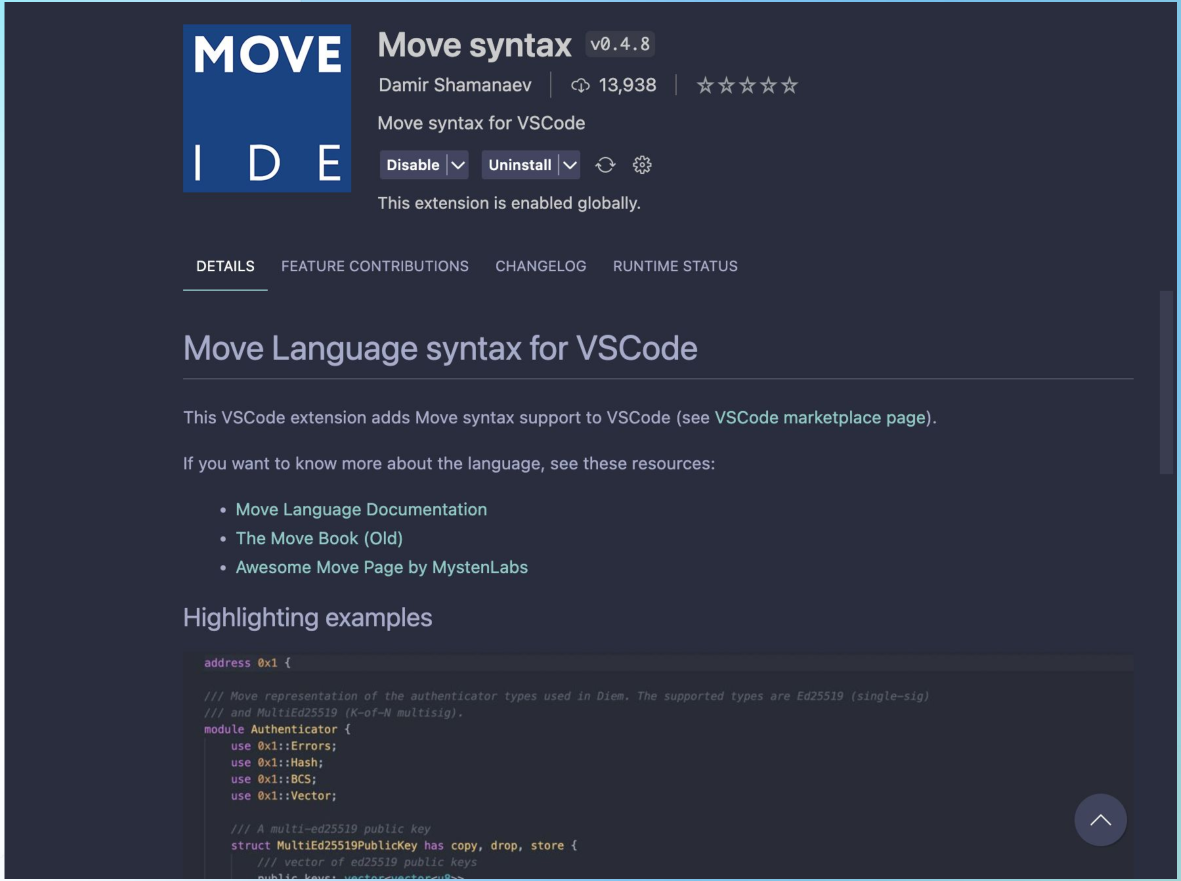
01 Ensure prerequisites, Sui binaries, IDE syntax highlighter are all installed

02 Create our first smart contract!

Install Sui

docs.sui.io/build/install

Install Move Syntax Plug-in



Move syntax v0.4.8

Damir Shamanaev | 13,938 | ☆☆☆☆☆

Move syntax for VSCode

This extension is enabled globally.

Move Language syntax for VSCode

This VSCode extension adds Move syntax support to VSCode (see VSCode marketplace page).

If you want to know more about the language, see these resources:

- Move Language Documentation
- The Move Book (Old)
- Awesome Move Page by MystenLabs

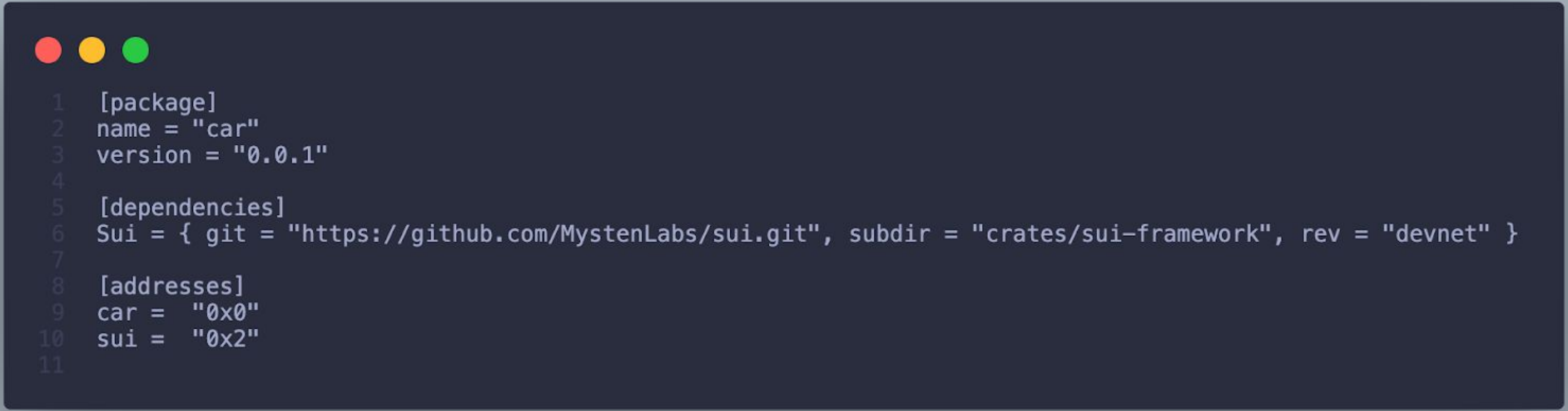
Highlighting examples

```
address 0x1 {

/// Move representation of the authenticator types used in Diem. The supported types are Ed25519 (single-sig)
/// and MultiEd25519 (K-of-N multisig).
module Authenticator {
    use 0x1::Errors;
    use 0x1::Hash;
    use 0x1::BCS;
    use 0x1::Vector;

    /// A multi-ed25519 public key
    struct MultiEd25519PublicKey has copy, drop, store {
        /// vector of ed25519 public keys
```

Our First Smart Contract!

In a new terminal, enter: `sui move new car`

sui

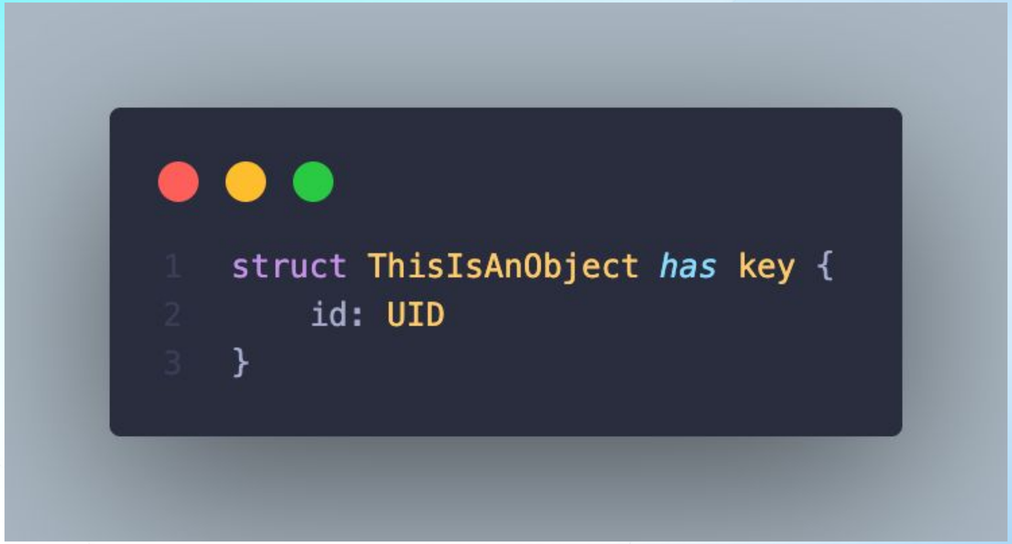


```toml
[package]
name = "car"
version = "0.0.1"

[dependencies]
Sui = { git = "https://github.com/MystenLabs/sui.git", subdir = "crates/sui-framework", rev = "devnet" }

[addresses]
car =  "0x0"
sui =  "0x2"
```

湾岸ミッドナイト
MIDNIGHT
MAXIMUM TUNE
6
いつでも4台バトル!
多人数分身対戦解禁!!
CAUTION
C1エリア C1内回り
深夜
Now Loading...
1PLAY ¥100
CONTINUE ¥100
¥300
CARD OUT
ARCADE
.TOKYO

sui

Objects 101

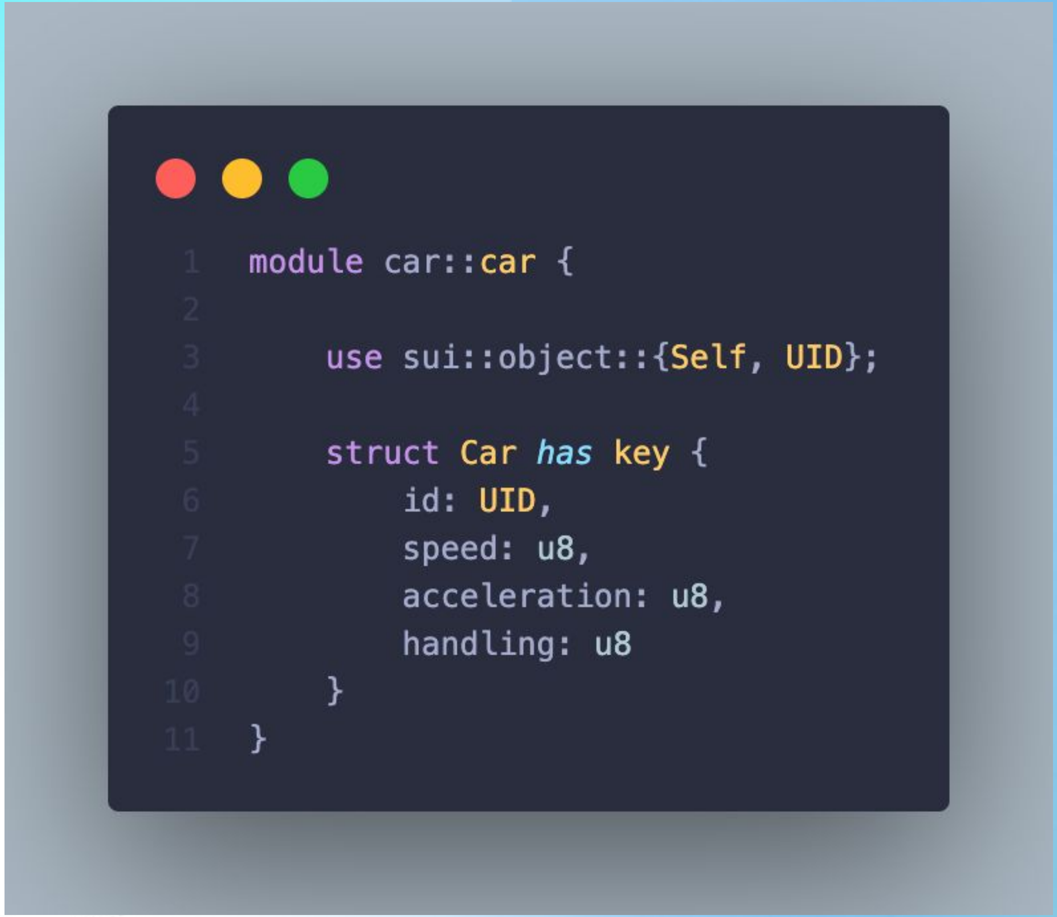


```
struct ThisIsAnObject has key {
    id: UID
}
```

Objects can be:

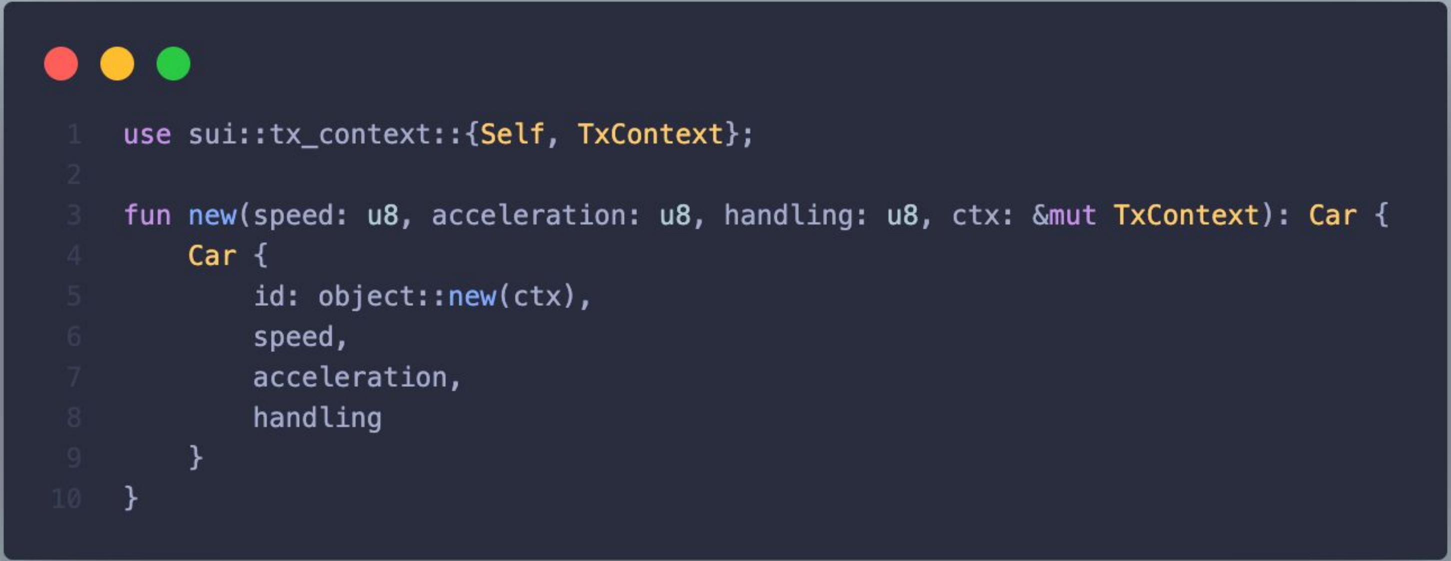
- Owned by an address
- Owned by another object
- Shared
- Immutable

sui



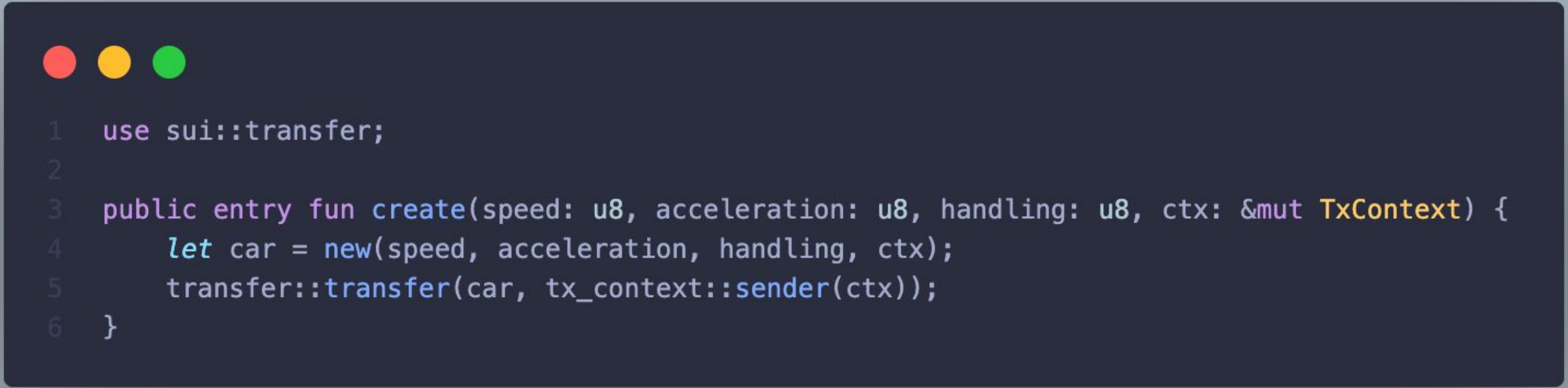
```
module car::car {

    use sui::object::{Self, UID};

    struct Car has key {
        id: UID,
        speed: u8,
        acceleration: u8,
        handling: u8
    }
}
```

sui

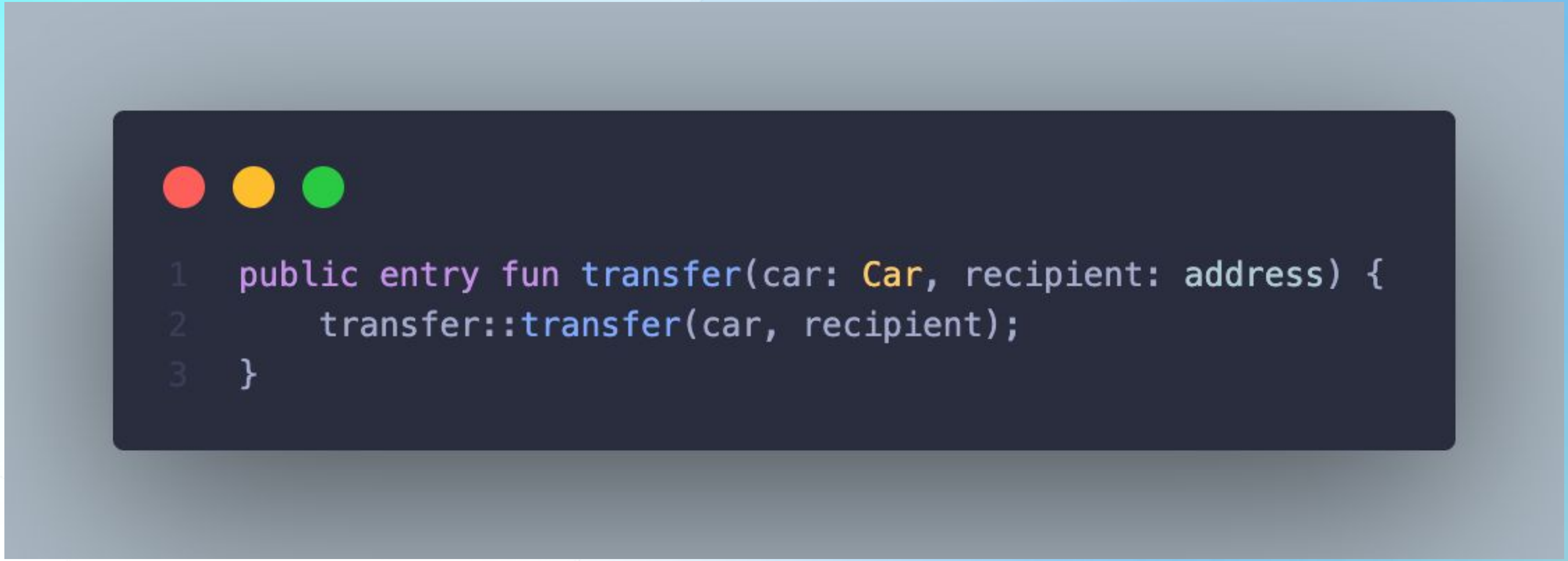


```
use sui::tx_context::{Self, TxContext};

fun new(speed: u8, acceleration: u8, handling: u8, ctx: &mut TxContext): Car {
    Car {
        id: object::new(ctx),
        speed,
        acceleration,
        handling
    }
}
```

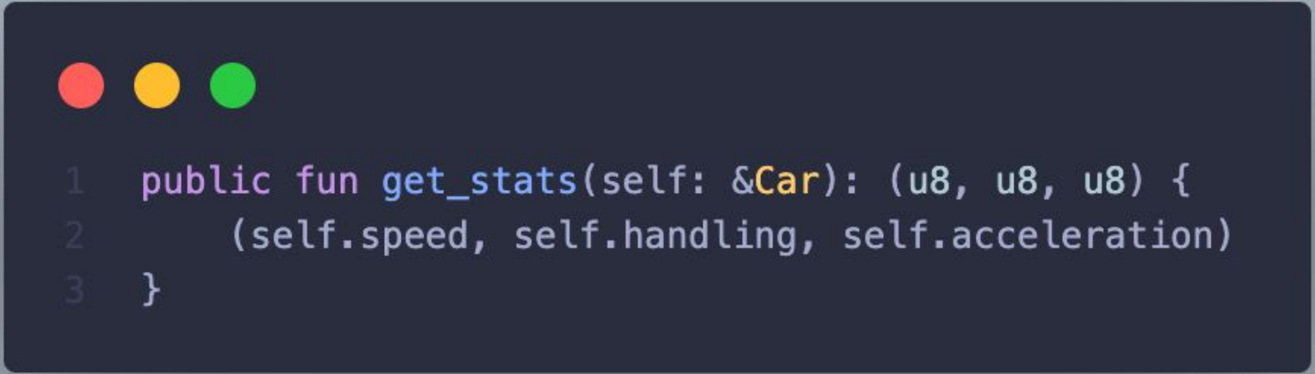
sui



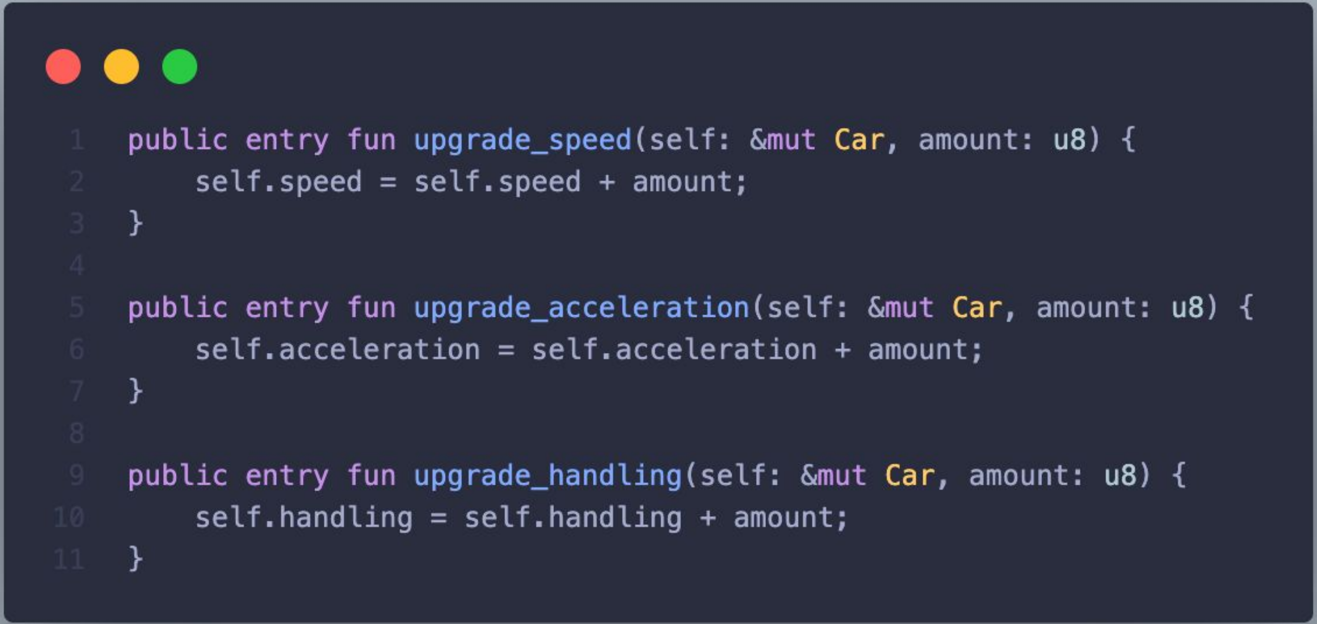
```
use sui::transfer;

public entry fun create(speed: u8, acceleration: u8, handling: u8, ctx: &mut TxContext) {
    let car = new(speed, acceleration, handling, ctx);
    transfer::transfer(car, tx_context::sender(ctx));
}
```



```
public entry fun transfer(car: Car, recipient: address) {
    transfer::transfer(car, recipient);
}
```



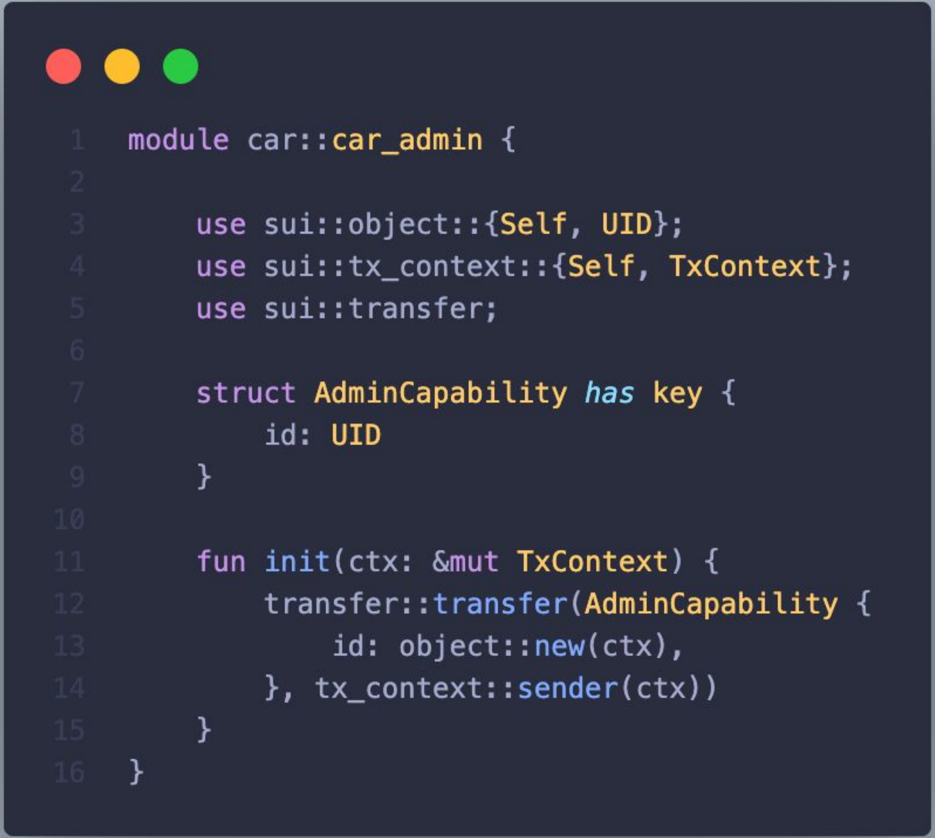
```
public fun get_stats(self: &Car): (u8, u8, u8) {
    (self.speed, self.handling, self.acceleration)
}
```



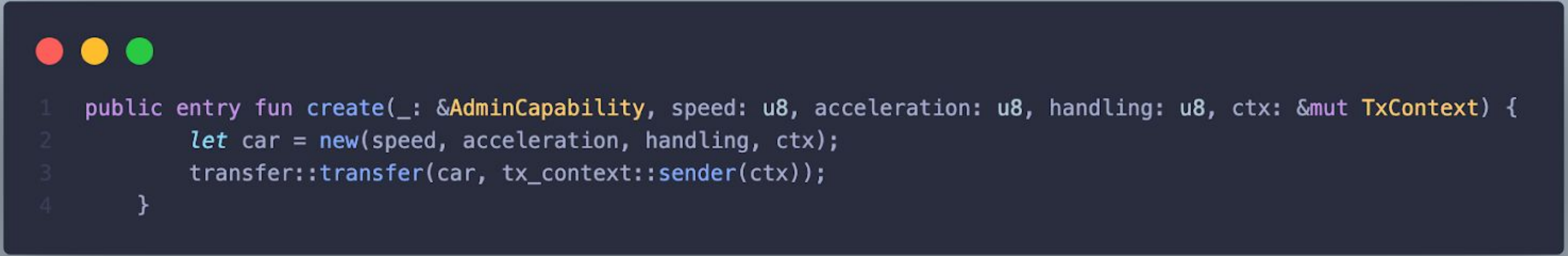
```
public entry fun upgrade_speed(self: &mut Car, amount: u8) {
    self.speed = self.speed + amount;
}

public entry fun upgrade_acceleration(self: &mut Car, amount: u8) {
    self.acceleration = self.acceleration + amount;
}

public entry fun upgrade_handling(self: &mut Car, amount: u8) {
    self.handling = self.handling + amount;
}
```

Recap:

- **Sui utilizes an object-centric programming model**
- **Objects represent ownership**



```
module car::car_admin {

    use sui::object::{Self, UID};
    use sui::tx_context::{Self, TxContext};
    use sui::transfer;

    struct AdminCapability has key {
        id: UID
    }

    fun init(ctx: &mut TxContext) {
        transfer::transfer(AdminCapability {
            id: object::new(ctx),
        }, tx_context::sender(ctx))
    }
}
```



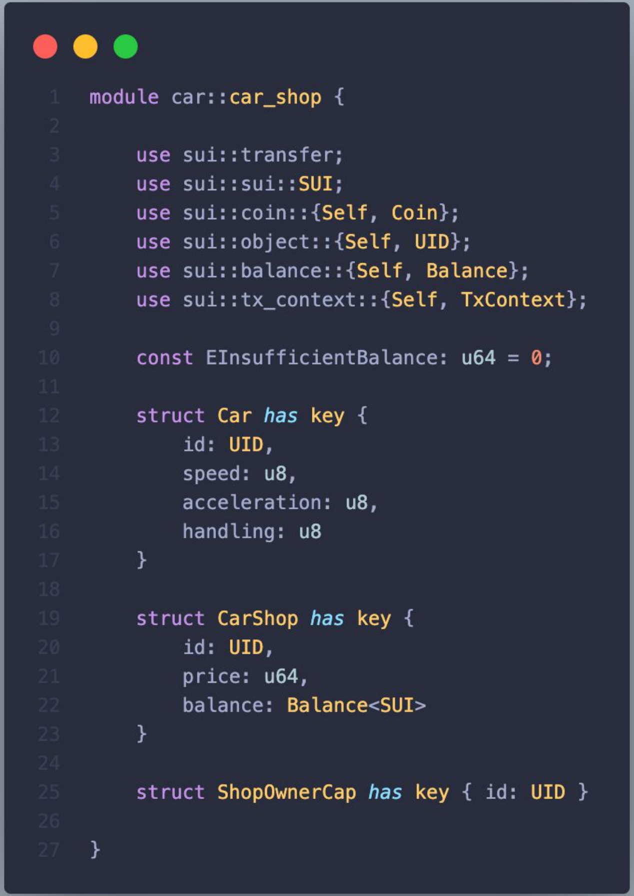
```
public entry fun create(_: &AdminCapability, speed: u8, acceleration: u8, handling: u8, ctx: &mut TxContext) {
        let car = new(speed, acceleration, handling, ctx);
        transfer::transfer(car, tx_context::sender(ctx));
    }
```

sui

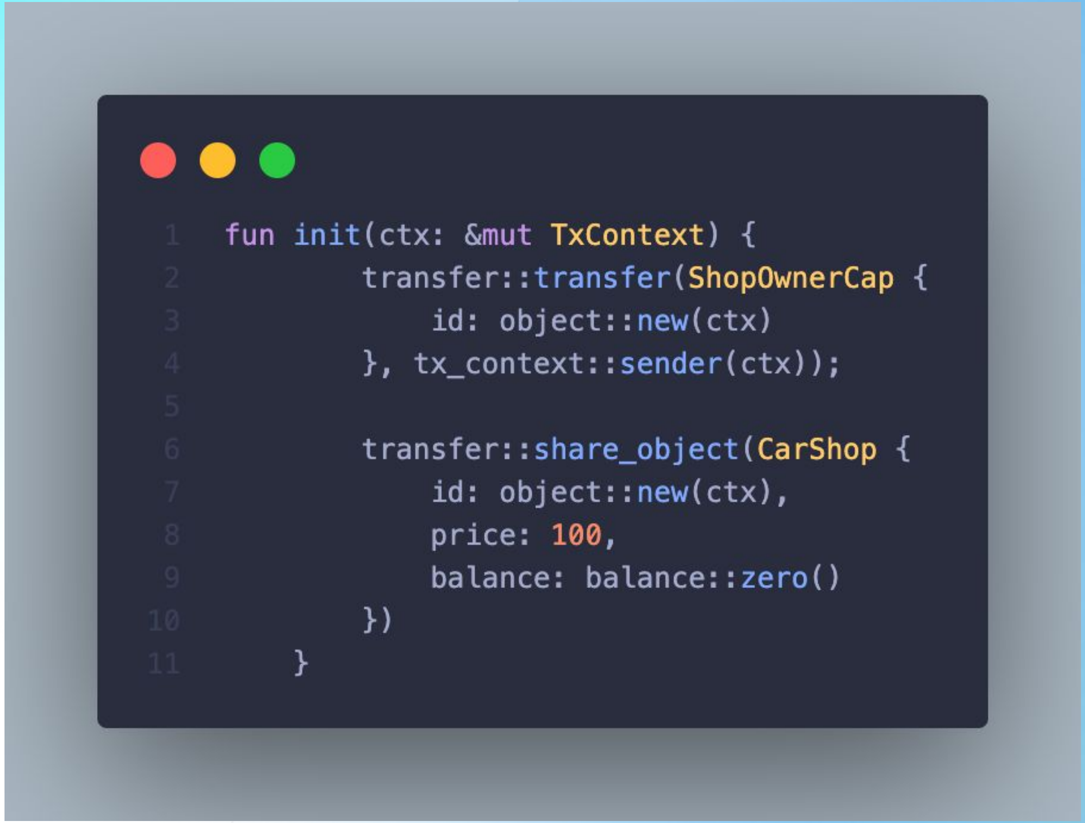
Recap:

- Capabilities can be used to gate admin access for functions

sui

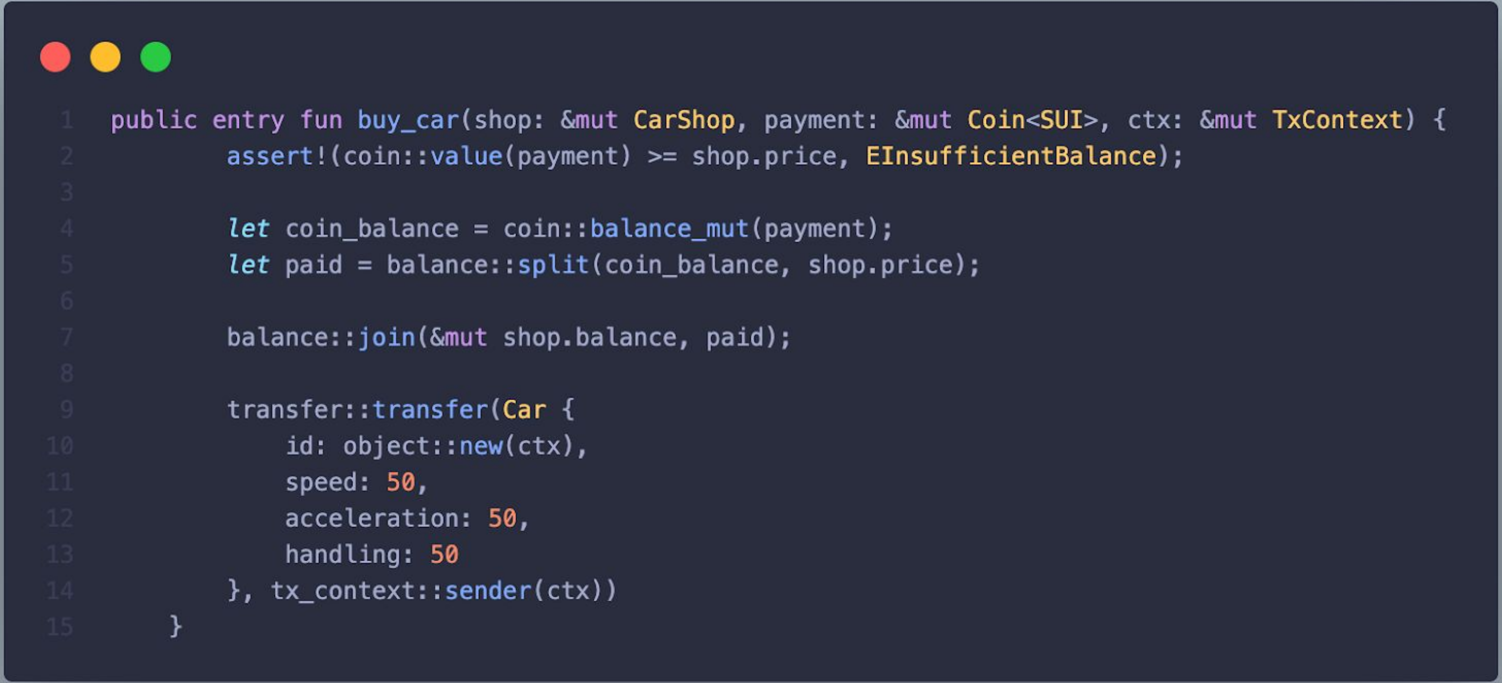

```
module car::car_shop {

    use sui::transfer;
    use sui::sui::SUI;
    use sui::coin::{Self, Coin};
    use sui::object::{Self, UID};
    use sui::balance::{Self, Balance};
    use sui::tx_context::{Self, TxContext};

    const EInsufficientBalance: u64 = 0;

    struct Car has key {
        id: UID,
        speed: u8,
        acceleration: u8,
        handling: u8
    }

    struct CarShop has key {
        id: UID,
        price: u64,
        balance: Balance<SUI>
    }

    struct ShopOwnerCap has key { id: UID }

}
```

sui



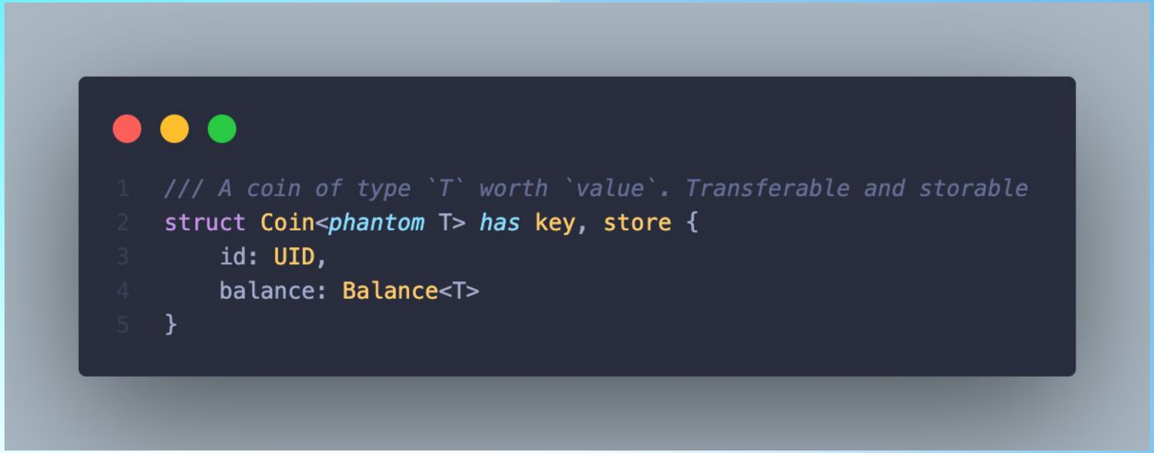
```
fun init(ctx: &mut TxContext) {
        transfer::transfer(ShopOwnerCap {
            id: object::new(ctx)
        }, tx_context::sender(ctx));

        transfer::share_object(CarShop {
            id: object::new(ctx),
            price: 100,
            balance: balance::zero()
        })
    }
```

sui

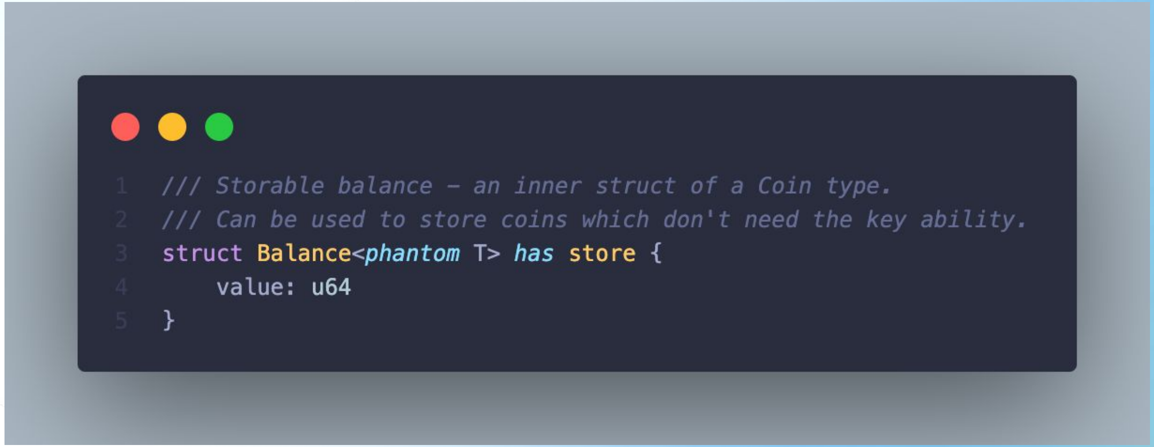


```
public entry fun buy_car(shop: &mut CarShop, payment: &mut Coin<SUI>, ctx: &mut TxContext) {
        assert!(coin::value(payment) >= shop.price, EInsufficientBalance);

        let coin_balance = coin::balance_mut(payment);
        let paid = balance::split(coin_balance, shop.price);

        balance::join(&mut shop.balance, paid);

        transfer::transfer(Car {
            id: object::new(ctx),
            speed: 50,
            acceleration: 50,
            handling: 50
        }, tx_context::sender(ctx))
    }
```

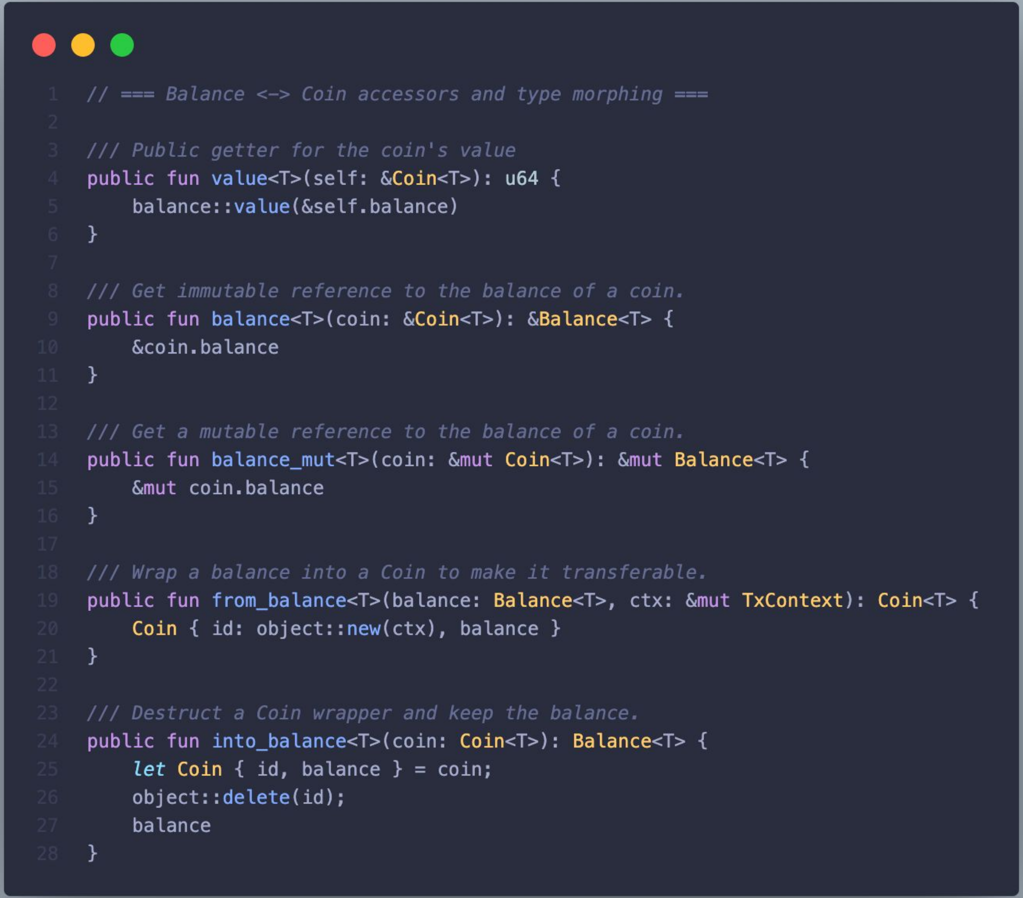
sui



```
/// A coin of type `T` worth `value`. Transferable and storable
struct Coin<phantom T> has key, store {
    id: UID,
    balance: Balance<T>
}
```

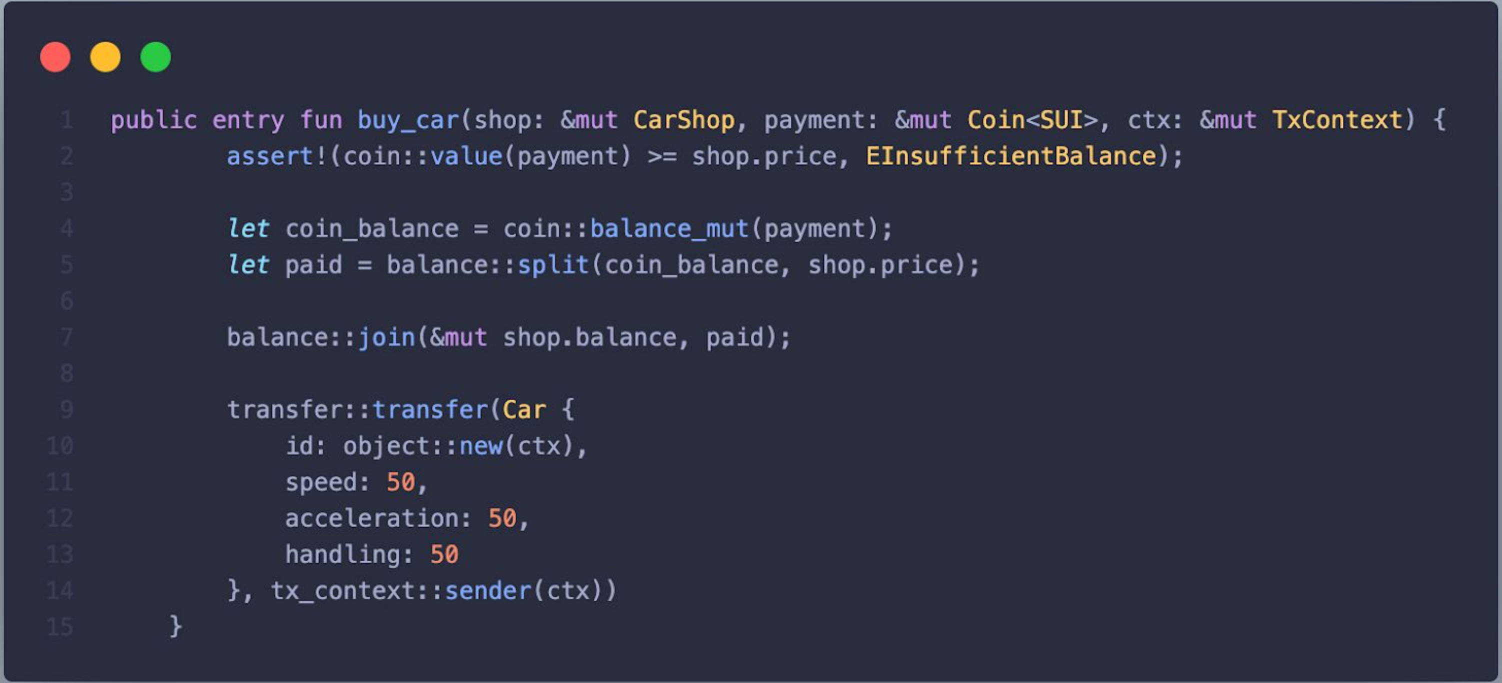


```
/// Storable balance - an inner struct of a Coin type.
/// Can be used to store coins which don't need the key ability.
struct Balance<phantom T> has store {
    value: u64
}
```



```
// === Balance <-> Coin accessors and type morphing ===

/// Public getter for the coin's value
public fun value<T>(self: &Coin<T>): u64 {
    balance::value(&self.balance)
}

/// Get immutable reference to the balance of a coin.
public fun balance<T>(coin: &Coin<T>): &Balance<T> {
    &coin.balance
}

/// Get a mutable reference to the balance of a coin.
public fun balance_mut<T>(coin: &mut Coin<T>): &mut Balance<T> {
    &mut coin.balance
}

/// Wrap a balance into a Coin to make it transferable.
public fun from_balance<T>(balance: Balance<T>, ctx: &mut TxContext): Coin<T> {
    Coin { id: object::new(ctx), balance }
}

/// Destruct a Coin wrapper and keep the balance.
public fun into_balance<T>(coin: Coin<T>): Balance<T> {
    let Coin { id, balance } = coin;
    object::delete(id);
    balance
}
```

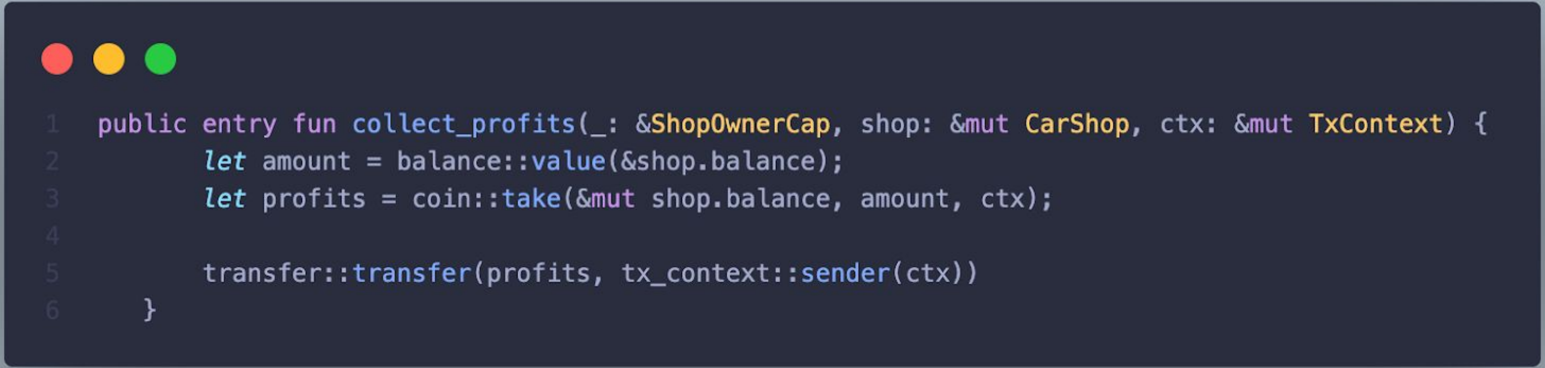
sui



```
public entry fun buy_car(shop: &mut CarShop, payment: &mut Coin<SUI>, ctx: &mut TxContext) {
        assert!(coin::value(payment) >= shop.price, EInsufficientBalance);

        let coin_balance = coin::balance_mut(payment);
        let paid = balance::split(coin_balance, shop.price);

        balance::join(&mut shop.balance, paid);

        transfer::transfer(Car {
            id: object::new(ctx),
            speed: 50,
            acceleration: 50,
            handling: 50
        }, tx_context::sender(ctx))
    }
```

sui

Object and Transaction Recap



```
public entry fun collect_profits(_: &ShopOwnerCap, shop: &mut CarShop, ctx: &mut TxContext) {
       let amount = balance::value(&shop.balance);
       let profits = coin::take(&mut shop.balance, amount, ctx);

       transfer::transfer(profits, tx_context::sender(ctx))
   }
```

sui

Recap:

- Shared objects can be accessed by anyone
- Interacting with shared objects is subject to consensus

sui

Bibliography/ Further Reading

docs.sui.io/learn

https://examples.sui.io/

What's Next!

sui

Next Workshop: Intro to **Sui Objects** and Creating Your First **NFT Project** in Sui Move

February 22, 2023

sui

Sui Denver Builder House!

lu.ma/suidenver

sui

Survey + Questions?

Twitter: @0xShayan

sui

sui