Writing your first Smart Contract

Advanced Move concepts, Sui-specific features, framework integration, and practical projects

Agenda

1. Advanced Move Concepts for Sui
2. Testing and Verification & Gas Optimization
3. Shared Objects, Dynamic Fields
4. Clock Object and Time, Decentralized Governance
5. Integration with Sui Framework and Libraries
6. Practical Exercises

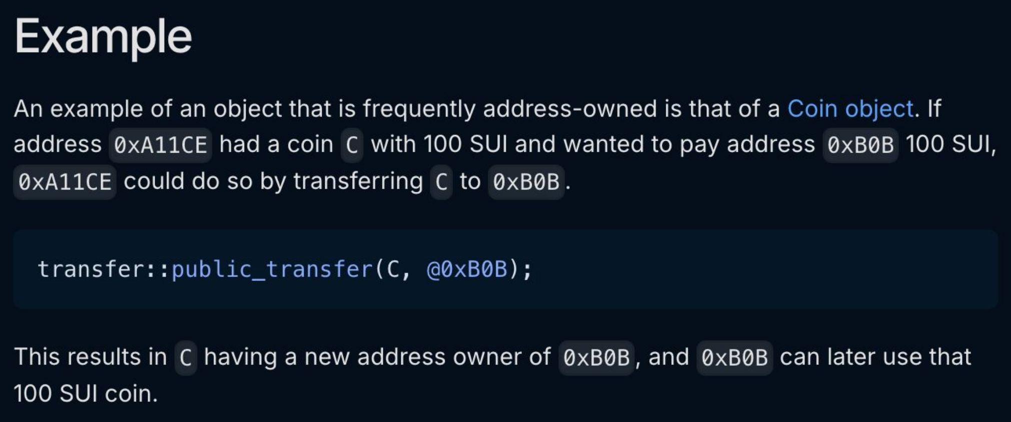
Advanced Move Concepts

Concept	Description	Relevance to Advanced Objects
Object-Centric Global Storage	Sui eliminates global storage, using objects with unique IDs for scalability and parallel processing	Central to Sui's architecture replacing traditional storage models
Addresses and Object IDs	Objects and accounts use 32-byte identifiers, with objects storing IDs in `id:UID` field	Ensure unique addressing, crucial for object management
Object with Key Ability	Requires `key` ability, with `id:UID` as the first field, enforced by bytecode verifier	Defines objects as first-class entities on-chain
Module Initializers	Special function run once at publication for setting up module-specific data, e.g., singleton objects	Facilitates initial object setup in smart contracts
Entry Functions	Callable in PTBs, used for atomic operations like on-chain randomness, with specific input restrictions	Enhances object interaction in complex transactions
Ownership Models	Includes exclusive (owner-only mutation), shared (anyone can mutate, needs consensus), and immutable (cannot change)	Critical for controlling object access and security
Dynamic Fields	Allow adding/removing fields at runtime, enabling flexible data structures	Advanced feature for extensible object design

Ownership Models

Details exclusive ownership (owner-only mutation), shared objects (mutable by anyone, requires consensus), and immutable objects (cannot change, used for packages).

Link: https://docs.sui.io/concepts/object-model



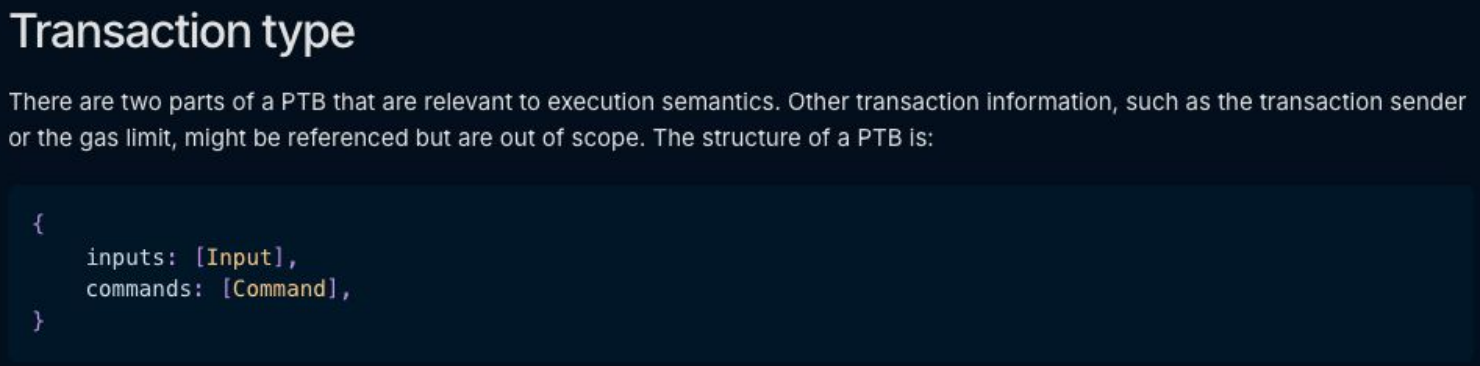
Example

An example of an object that is frequently address-owned is that of a Coin object. If address `0xA11CE` had a coin `C` with 100 SUI and wanted to pay address `0xB0B` 100 SUI, `0xA11CE` could do so by transferring `C` to `0xB0B`.

```
transfer::public_transfer(C, @0xB0B);
```

This results in `C` having a new address owner of `0xB0B`, and `0xB0B` can later use that 100 SUI coin.

Advanced Features: PTBs



Transaction type

There are two parts of a PTB that are relevant to execution semantics. Other transaction information, such as the transaction sender or the gas limit, might be referenced but are out of scope. The structure of a PTB is:

```
{
    inputs: [Input],
    commands: [Command],
}
```

- The inputs value is a vector of arguments
- The commands value is a vector of commands, and the possible commands are: TransferObjects, SplitCoins, MakeMoveVec, ...

https://docs.sui.io/concepts/transactions/prog-txn-blocks

Testing, Verification & Gas Optimization

Testing

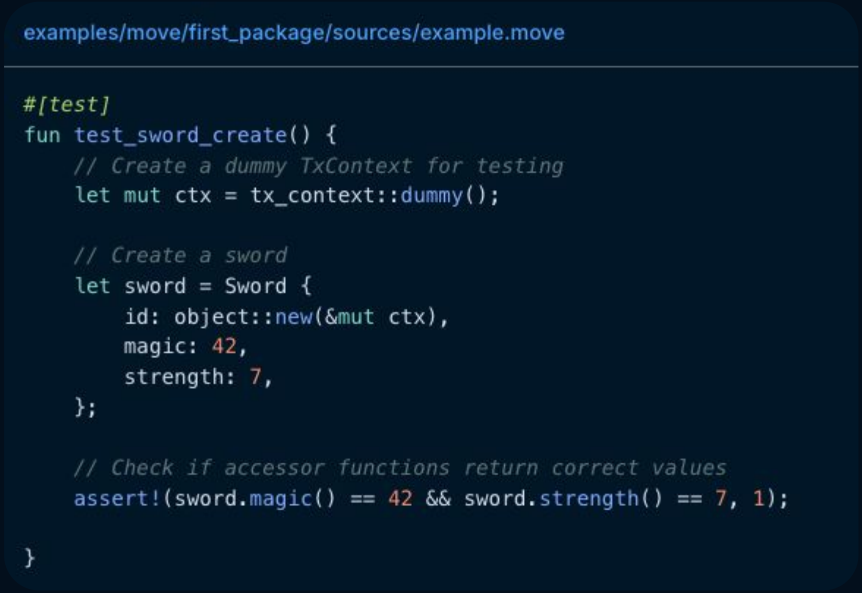
Write unit tests using #[test] attribute to ensure correctness.

Example: assert!(2 + 2 == 4);

Using SUI to command to run test:

```
sui move test
```

⇒ Testing guide | Sample Code



examples/move/first_package/sources/example.move

```
#[test]
fun test_sword_create() {
    // Create a dummy TxContext for testing
    let mut ctx = tx_context::dummy();

    // Create a sword
    let sword = Sword {
        id: object::new(&mut ctx),
        magic: 42,
        strength: 7,
    };

    // Check if accessor functions return correct values
    assert!(sword.magic() == 42 && sword.strength() == 7, 1);

}
```

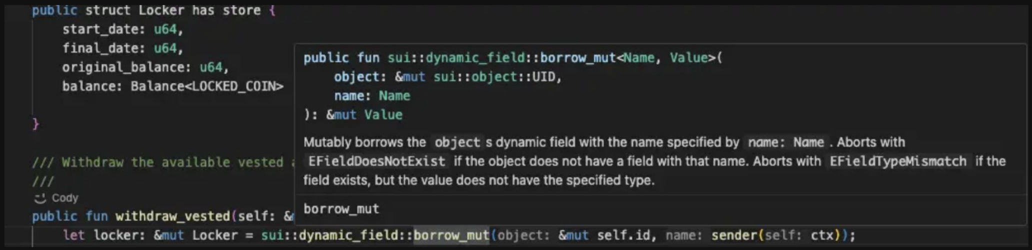
Testing, Verification & Gas Optimization

Verification

Use Move Prover for formal verification of smart contracts.

*Example: Prove functions abort under specific conditions.

<u>Sui-move-analyzer</u>



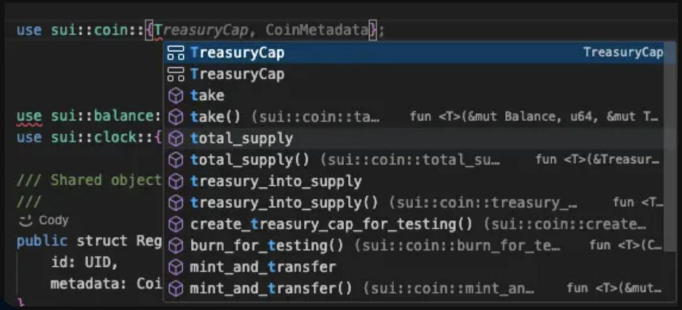
```
public struct Locker has store {
    start_date: u64,
    final_date: u64,
    original_balance: u64,
    balance: Balance<LOCKED_COIN>

}

/// Withdraw the available vested a
///
Cody
public fun withdraw_vested(self: &
    let locker: &mut Locker = sui::dynamic_field::borrow_mut(object: &mut self.id, name: sender(self: ctx));
```

```
public fun sui::dynamic_field::borrow_mut<Name, Value>(
    object: &mut sui::object::UID,
    name: Name
): &mut Value
```

Mutably borrows the `object` s dynamic field with the name specified by `name: Name`. Aborts with `EFieldDoesNotExist` if the object does not have a field with that name. Aborts with `EFieldTypeMismatch` if the field exists, but the value does not have the specified type.

`borrow_mut`

- Auto complete các modules, functions, fields, structs, etc:



```
use sui::coin::{TreasuryCap, CoinMetadata};


use sui::balance:
use sui::clock::{

/// Shared object
///
Cody
public struct Reg
    id: UID,
    metadata: Coi
}
```

```
TreasuryCap                                                  TreasuryCap
TreasuryCap
take
take() (sui::coin::ta…          fun <T>(&mut Balance, u64, &mut T…
total_supply
total_supply() (sui::coin::total_su…           fun <T>(&Treasur…
treasury_into_supply
treasury_into_supply() (sui::coin::treasury_…        fun <T…
create_treasury_cap_for_testing() (sui::coin::create…
burn_for_testing() (sui::coin::burn_for_te…        fun <T>(C…
mint_and_transfer
mint_and_transfer() (sui::coin::mint_an…        fun <T>(&mut…
```

Testing, Verification & Gas Optimization

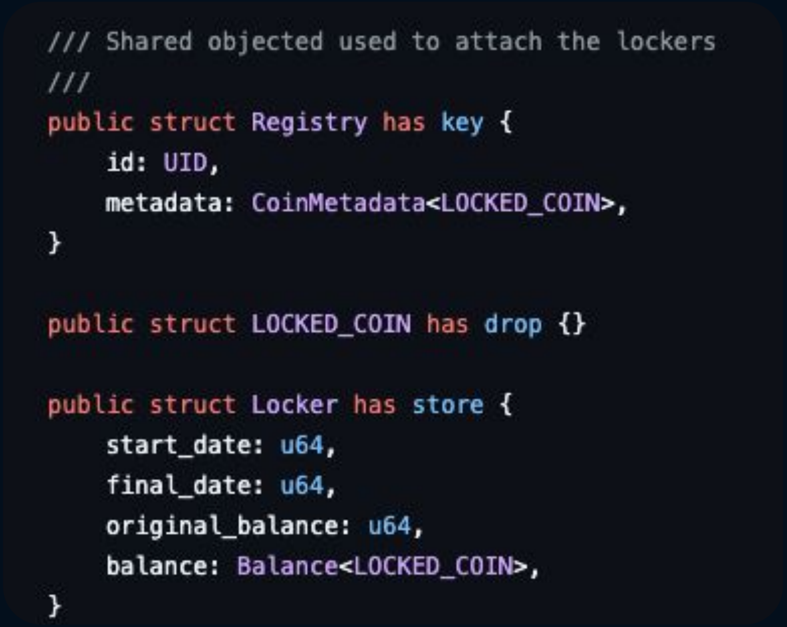
Gas Optimization

Minimize gas costs by reducing storage usage and computation.

Verify gas budget for smart contract function:

https://docs.sui.io/concepts/tokenomics/gas-in-sui

⇒ Sui Gas Model



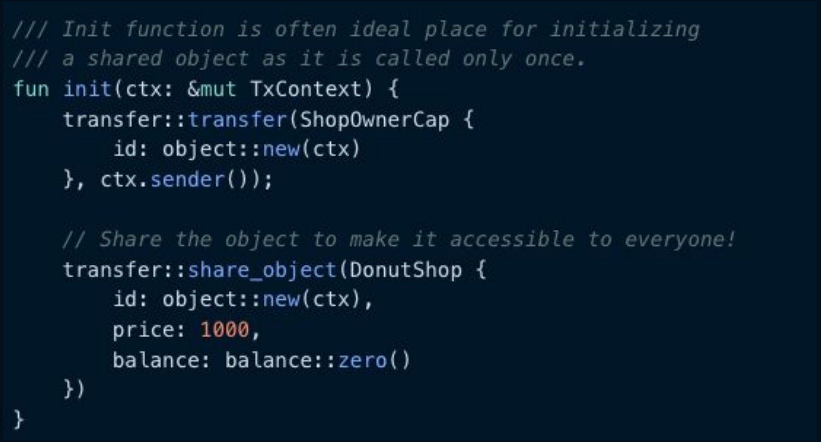
```
/// Shared objected used to attach the lockers
///
public struct Registry has key {
    id: UID,
    metadata: CoinMetadata<LOCKED_COIN>,
}

public struct LOCKED_COIN has drop {}

public struct Locker has store {
    start_date: u64,
    final_date: u64,
    original_balance: u64,
    balance: Balance<LOCKED_COIN>,
}
```

Shared Objects, Dynamic Fields

Shared Objects

Objects accessible and mutable by multiple parties for collaboration.

*Example: Shared object in a decentralized exchange (DEX).



```
/// Init function is often ideal place for initializing
/// a shared object as it is called only once.
fun init(ctx: &mut TxContext) {
    transfer::transfer(ShopOwnerCap {
        id: object::new(ctx)
    }, ctx.sender());

    // Share the object to make it accessible to everyone!
    transfer::share_object(DonutShop {
        id: object::new(ctx),
        price: 1000,
        balance: balance::zero()
    })
}
```

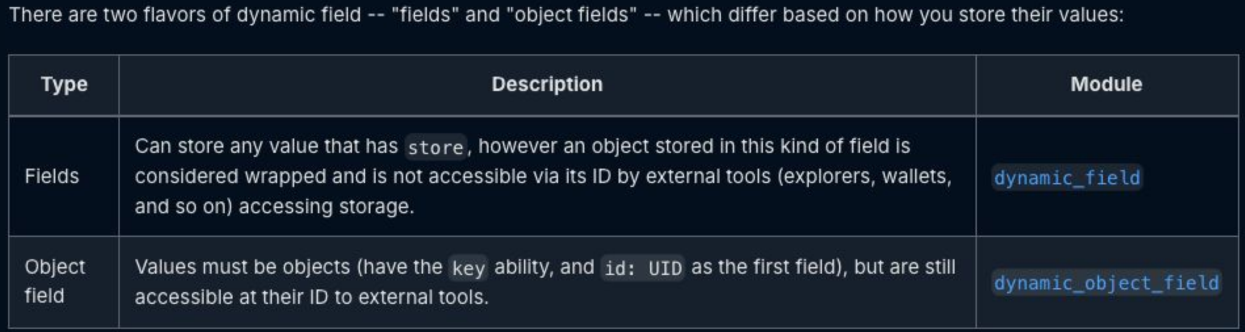
https://docs.sui.io/concepts/object-ownership/shared

Shared Objects, Dynamic Fields

Dynamic Fields

Add fields to objects at runtime for flexible data structures.

*Example: Dynamically adding metadata to NFTs.



There are two flavors of dynamic field -- "fields" and "object fields" -- which differ based on how you store their values:

Type	Description	Module
Fields	Can store any value that has `store`, however an object stored in this kind of field is considered wrapped and is not accessible via its ID by external tools (explorers, wallets, and so on) accessing storage.	`dynamic_field`
Object field	Values must be objects (have the `key` ability, and `id: UID` as the first field), but are still accessible at their ID to external tools.	`dynamic_object_field`

https://docs.sui.io/concepts/dynamic-fields

Clock Object and Time, Decentralized Governance

Clock Object and Time

✓ Access on-chain time for time-based smart contract logic.

*Example: Use sui::clock::timestamp_ms for vesting schedules.

Decentralized Governance

✓ Implement governance using capabilities and shared objects.

*Example: Voting on protocol upgrades via SIPs (Sui Improvement Proposals).

Time access | SIPs repository

Integration with Sui Framework and Libraries

Sui Framework

Use standard Sui modules for streamlined development.

*Example: sui::coin for custom token implementations.

Learn more

Libraries

Explore standard libraries for common functionalities.

*Example: sui::vector for dynamic arrays.

Learn more

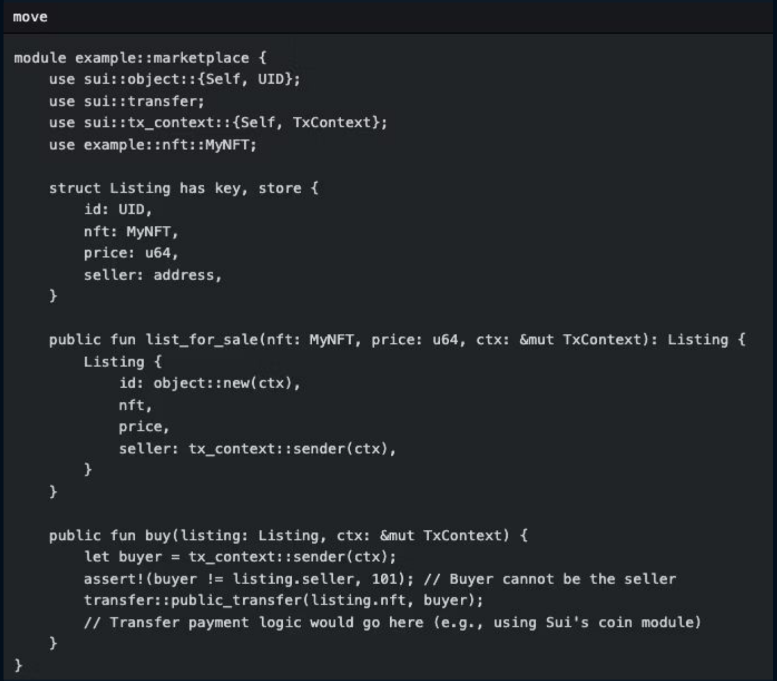
Practical Exercises

Creating an NFT

Define an NFT struct with metadata like name and description.

Marketplace Logic

Functions for listing NFTs and enabling purchases using Listing resources.



move

```move
module example::marketplace {
    use sui::object::{Self, UID};
    use sui::transfer;
    use sui::tx_context::{Self, TxContext};
    use example::nft::MyNFT;

    struct Listing has key, store {
        id: UID,
        nft: MyNFT,
        price: u64,
        seller: address,
    }

    public fun list_for_sale(nft: MyNFT, price: u64, ctx: &mut TxContext): Listing {
        Listing {
            id: object::new(ctx),
            nft,
            price,
            seller: tx_context::sender(ctx),
        }
    }

    public fun buy(listing: Listing, ctx: &mut TxContext) {
        let buyer = tx_context::sender(ctx);
        assert!(buyer != listing.seller, 101); // Buyer cannot be the seller
        transfer::public_transfer(listing.nft, buyer);
        // Transfer payment logic would go here (e.g., using Sui's coin module)
    }
}
```

Thank You.