[](https://ida.interchain.io/)

[Interchain Developer Academy](https://ida.interchain.io/)/[Interchain Developer Academy](https://ida.interchain.io/hands-on-exercise/2-ignite-cli-adv/6-payment-winning.html)



Search

[Interchain Developer Academy](https://ida.interchain.io/)[Interchain Developer Academy](https://ida.interchain.io/hands-on-exercise/2-ignite-cli-adv/6-payment-winning.html)

Search



Filters

Interchain Developer Academy

[](https://ida.interchain.io/hands-on-exercise/2-ignite-cli-adv/6-payment-winning.html)

[Week 0 - Getting Started](https://ida.interchain.io/hands-on-exercise/2-ignite-cli-adv/6-payment-winning.html)

[Getting Started](https://ida.interchain.io/hands-on-exercise/2-ignite-cli-adv/6-payment-winning.html)

[Blockchain 101](https://ida.interchain.io/hands-on-exercise/2-ignite-cli-adv/6-payment-winning.html)

[Blockchain History](https://ida.interchain.io/hands-on-exercise/2-ignite-cli-adv/6-payment-winning.html)

[Public and Managed Blockchains](https://ida.interchain.io/hands-on-exercise/2-ignite-cli-adv/6-payment-winning.html)

[Consensus in Distributed Networks](https://ida.interchain.io/hands-on-exercise/2-ignite-cli-adv/6-payment-winning.html)

[Cryptography](https://ida.interchain.io/hands-on-exercise/2-ignite-cli-adv/6-payment-winning.html)

[Self-Assessment Quiz](https://ida.interchain.io/hands-on-exercise/2-ignite-cli-adv/6-payment-winning.html)

[Go Introduction - First Steps](https://ida.interchain.io/hands-on-exercise/2-ignite-cli-adv/6-payment-winning.html)

[Go Basics](https://ida.interchain.io/hands-on-exercise/2-ignite-cli-adv/6-payment-winning.html)

[Go Interfaces](https://ida.interchain.io/hands-on-exercise/2-ignite-cli-adv/6-payment-winning.html)

[Control Structures in Go](https://ida.interchain.io/hands-on-exercise/2-ignite-cli-adv/6-payment-winning.html)

[Arrays and Slices in Go](https://ida.interchain.io/hands-on-exercise/2-ignite-cli-adv/6-payment-winning.html)

[Standard Packages in Go](https://ida.interchain.io/hands-on-exercise/2-ignite-cli-adv/6-payment-winning.html)

[Concurrency in Go](https://ida.interchain.io/hands-on-exercise/2-ignite-cli-adv/6-payment-winning.html)

[Good-To-Know Dev Terms](https://ida.interchain.io/hands-on-exercise/2-ignite-cli-adv/6-payment-winning.html)

[Docker Introduction](https://ida.interchain.io/hands-on-exercise/2-ignite-cli-adv/6-payment-winning.html)

[](https://ida.interchain.io/hands-on-exercise/2-ignite-cli-adv/6-payment-winning.html)

[Week 1 - Introduction to the Interchain](https://ida.interchain.io/hands-on-exercise/2-ignite-cli-adv/6-payment-winning.html)

[Introduction to the Interchain](https://ida.interchain.io/hands-on-exercise/2-ignite-cli-adv/6-payment-winning.html)

[Blockchain Technology and the Interchain](https://ida.interchain.io/hands-on-exercise/2-ignite-cli-adv/6-payment-winning.html)

[The Interchain Ecosystem](https://ida.interchain.io/hands-on-exercise/2-ignite-cli-adv/6-payment-winning.html)

[Getting ATOM and Staking It](https://ida.interchain.io/hands-on-exercise/2-ignite-cli-adv/6-payment-winning.html)

[A Blockchain App Architecture](https://ida.interchain.io/hands-on-exercise/2-ignite-cli-adv/6-payment-winning.html)

[Accounts](https://ida.interchain.io/hands-on-exercise/2-ignite-cli-adv/6-payment-winning.html)

[Transactions](https://ida.interchain.io/hands-on-exercise/2-ignite-cli-adv/6-payment-winning.html)

[Messages](https://ida.interchain.io/hands-on-exercise/2-ignite-cli-adv/6-payment-winning.html)

[Modules](https://ida.interchain.io/hands-on-exercise/2-ignite-cli-adv/6-payment-winning.html)

[Protobuf](https://ida.interchain.io/hands-on-exercise/2-ignite-cli-adv/6-payment-winning.html)

[Multistore and Keepers](https://ida.interchain.io/hands-on-exercise/2-ignite-cli-adv/6-payment-winning.html)

[BaseApp](https://ida.interchain.io/hands-on-exercise/2-ignite-cli-adv/6-payment-winning.html)

[Queries](https://ida.interchain.io/hands-on-exercise/2-ignite-cli-adv/6-payment-winning.html)

[Events](https://ida.interchain.io/hands-on-exercise/2-ignite-cli-adv/6-payment-winning.html)

[Context](https://ida.interchain.io/hands-on-exercise/2-ignite-cli-adv/6-payment-winning.html)

[Testing](https://ida.interchain.io/hands-on-exercise/2-ignite-cli-adv/6-payment-winning.html)

[Relaying with IBC](https://ida.interchain.io/hands-on-exercise/2-ignite-cli-adv/6-payment-winning.html)

[Interchain Security](https://ida.interchain.io/hands-on-exercise/2-ignite-cli-adv/6-payment-winning.html)

[Bridges](https://ida.interchain.io/hands-on-exercise/2-ignite-cli-adv/6-payment-winning.html)

[Migrations](https://ida.interchain.io/hands-on-exercise/2-ignite-cli-adv/6-payment-winning.html)

[Week 1 Quiz](https://ida.interchain.io/hands-on-exercise/2-ignite-cli-adv/6-payment-winning.html)

[](https://ida.interchain.io/hands-on-exercise/2-ignite-cli-adv/6-payment-winning.html)

[Week 2 - First Steps](https://ida.interchain.io/hands-on-exercise/2-ignite-cli-adv/6-payment-winning.html)

[First Steps](https://ida.interchain.io/hands-on-exercise/2-ignite-cli-adv/6-payment-winning.html)

[Setup Your Work Environment](https://ida.interchain.io/hands-on-exercise/2-ignite-cli-adv/6-payment-winning.html)

[Run a Node, API, and CLI](https://ida.interchain.io/hands-on-exercise/2-ignite-cli-adv/6-payment-winning.html)

[Ignite CLI](https://ida.interchain.io/hands-on-exercise/2-ignite-cli-adv/6-payment-winning.html)

[Exercise - Make a Checkers Blockchain](https://ida.interchain.io/hands-on-exercise/2-ignite-cli-adv/6-payment-winning.html)

[Store Object](https://ida.interchain.io/hands-on-exercise/2-ignite-cli-adv/6-payment-winning.html)

[Create Custom Messages](https://ida.interchain.io/hands-on-exercise/2-ignite-cli-adv/6-payment-winning.html)

[Create and Save a Game Properly](https://ida.interchain.io/hands-on-exercise/2-ignite-cli-adv/6-payment-winning.html)

[Add a Way to Make a Move](https://ida.interchain.io/hands-on-exercise/2-ignite-cli-adv/6-payment-winning.html)

[Emit Game Information](https://ida.interchain.io/hands-on-exercise/2-ignite-cli-adv/6-payment-winning.html)

[Record the Game Winner](https://ida.interchain.io/hands-on-exercise/2-ignite-cli-adv/6-payment-winning.html)

[Week 2 Exercise](https://ida.interchain.io/hands-on-exercise/2-ignite-cli-adv/6-payment-winning.html)

[](https://ida.interchain.io/hands-on-exercise/2-ignite-cli-adv/6-payment-winning.html)

[Week 3 - Introduction to IBC and CosmJS](https://ida.interchain.io/hands-on-exercise/2-ignite-cli-adv/6-payment-winning.html)

[Introduction to IBC and CosmJS](https://ida.interchain.io/hands-on-exercise/2-ignite-cli-adv/6-payment-winning.html)

[What is IBC?](https://ida.interchain.io/hands-on-exercise/2-ignite-cli-adv/6-payment-winning.html)

[IBC/TAO - Connections (OPTIONAL)](https://ida.interchain.io/hands-on-exercise/2-ignite-cli-adv/6-payment-winning.html)

[IBC/TAO - Channels (OPTIONAL)](https://ida.interchain.io/hands-on-exercise/2-ignite-cli-adv/6-payment-winning.html)

[IBC/TAO - Clients (OPTIONAL)](https://ida.interchain.io/hands-on-exercise/2-ignite-cli-adv/6-payment-winning.html)

[IBC Token Transfer](https://ida.interchain.io/hands-on-exercise/2-ignite-cli-adv/6-payment-winning.html)

[Interchain Accounts (OPTIONAL)](https://ida.interchain.io/hands-on-exercise/2-ignite-cli-adv/6-payment-winning.html)

[IBC Middleware (OPTIONAL)](https://ida.interchain.io/hands-on-exercise/2-ignite-cli-adv/6-payment-winning.html)

[Create a Custom IBC Middleware (OPTIONAL)](https://ida.interchain.io/hands-on-exercise/2-ignite-cli-adv/6-payment-winning.html)

[Integrate IBC Middleware Into a Chain (OPTIONAL)](https://ida.interchain.io/hands-on-exercise/2-ignite-cli-adv/6-payment-winning.html)

[IBC Tooling](https://ida.interchain.io/hands-on-exercise/2-ignite-cli-adv/6-payment-winning.html)

[What is CosmJS?](https://ida.interchain.io/hands-on-exercise/2-ignite-cli-adv/6-payment-winning.html)

[Your First CosmJS Actions](https://ida.interchain.io/hands-on-exercise/2-ignite-cli-adv/6-payment-winning.html)

[Compose Complex Transactions](https://ida.interchain.io/hands-on-exercise/2-ignite-cli-adv/6-payment-winning.html)

[Learn to Integrate Keplr](https://ida.interchain.io/hands-on-exercise/2-ignite-cli-adv/6-payment-winning.html)

[Create Custom CosmJS Interfaces](https://ida.interchain.io/hands-on-exercise/2-ignite-cli-adv/6-payment-winning.html)

[](https://ida.interchain.io/hands-on-exercise/2-ignite-cli-adv/6-payment-winning.html)

[Week 4 - Ignite CLI and IBC Advanced](https://ida.interchain.io/hands-on-exercise/2-ignite-cli-adv/6-payment-winning.html)

[Ignite CLI and IBC Advanced](https://ida.interchain.io/hands-on-exercise/2-ignite-cli-adv/6-payment-winning.html)

[Keep an Up-To-Date Game Deadline](https://ida.interchain.io/hands-on-exercise/2-ignite-cli-adv/6-payment-winning.html)

[Keep Track Of How Many Moves Have Been Played](https://ida.interchain.io/hands-on-exercise/2-ignite-cli-adv/6-payment-winning.html)

[Put Your Games in Order](https://ida.interchain.io/hands-on-exercise/2-ignite-cli-adv/6-payment-winning.html)

[Auto-Expiring Games](https://ida.interchain.io/hands-on-exercise/2-ignite-cli-adv/6-payment-winning.html)

[Let Players Set a Wager](https://ida.interchain.io/hands-on-exercise/2-ignite-cli-adv/6-payment-winning.html)

[Handle wager payments](https://ida.interchain.io/hands-on-exercise/2-ignite-cli-adv/6-payment-winning.html)

[Integration tests](https://ida.interchain.io/hands-on-exercise/2-ignite-cli-adv/6-payment-winning.html)

[Incentivize Players](https://ida.interchain.io/hands-on-exercise/2-ignite-cli-adv/6-payment-winning.html)

[Help Find a Correct Move](https://ida.interchain.io/hands-on-exercise/2-ignite-cli-adv/6-payment-winning.html)

[Play With Cross-Chain Tokens](https://ida.interchain.io/hands-on-exercise/2-ignite-cli-adv/6-payment-winning.html)

[Understand IBC Denoms](https://ida.interchain.io/hands-on-exercise/2-ignite-cli-adv/6-payment-winning.html)

[Go Relayer](https://ida.interchain.io/hands-on-exercise/2-ignite-cli-adv/6-payment-winning.html)

[Hermes Relayer](https://ida.interchain.io/hands-on-exercise/2-ignite-cli-adv/6-payment-winning.html)

[](https://ida.interchain.io/hands-on-exercise/2-ignite-cli-adv/6-payment-winning.html)

[Week 5 - CosmJS Advanced](https://ida.interchain.io/hands-on-exercise/2-ignite-cli-adv/6-payment-winning.html)

[CosmJS Advanced](https://ida.interchain.io/hands-on-exercise/2-ignite-cli-adv/6-payment-winning.html)

[Create Custom Objects](https://ida.interchain.io/hands-on-exercise/2-ignite-cli-adv/6-payment-winning.html)

[Create Custom Messages](https://ida.interchain.io/hands-on-exercise/2-ignite-cli-adv/6-payment-winning.html)

[Get an External GUI](https://ida.interchain.io/hands-on-exercise/2-ignite-cli-adv/6-payment-winning.html)

[Integrate CosmJS and Keplr](https://ida.interchain.io/hands-on-exercise/2-ignite-cli-adv/6-payment-winning.html)

[Backend Script for Game Indexing](https://ida.interchain.io/hands-on-exercise/2-ignite-cli-adv/6-payment-winning.html)

[](https://ida.interchain.io/hands-on-exercise/2-ignite-cli-adv/6-payment-winning.html)

[Week 6 - IBC Deep Dive](https://ida.interchain.io/hands-on-exercise/2-ignite-cli-adv/6-payment-winning.html)

[IBC Deep Dive](https://ida.interchain.io/hands-on-exercise/2-ignite-cli-adv/6-payment-winning.html)

[IBC Application Developer Introduction](https://ida.interchain.io/hands-on-exercise/2-ignite-cli-adv/6-payment-winning.html)

[Make a Module IBC-Enabled](https://ida.interchain.io/hands-on-exercise/2-ignite-cli-adv/6-payment-winning.html)

[Adding Packet and Acknowledgment Data](https://ida.interchain.io/hands-on-exercise/2-ignite-cli-adv/6-payment-winning.html)

[Extend the Checkers Game With a Leaderboard](https://ida.interchain.io/hands-on-exercise/2-ignite-cli-adv/6-payment-winning.html)

[Create a Leaderboard Chain](https://ida.interchain.io/hands-on-exercise/2-ignite-cli-adv/6-payment-winning.html)

[](https://ida.interchain.io/hands-on-exercise/2-ignite-cli-adv/6-payment-winning.html)

[Week 7 - From Code to MVP to Production and Migrations](https://ida.interchain.io/hands-on-exercise/2-ignite-cli-adv/6-payment-winning.html)

[From Code to MVP to Production and Migrations](https://ida.interchain.io/hands-on-exercise/2-ignite-cli-adv/6-payment-winning.html)

[Run in Production](https://ida.interchain.io/hands-on-exercise/2-ignite-cli-adv/6-payment-winning.html)

[Prepare the Software to Run](https://ida.interchain.io/hands-on-exercise/2-ignite-cli-adv/6-payment-winning.html)

[Prepare a Validator and Keys](https://ida.interchain.io/hands-on-exercise/2-ignite-cli-adv/6-payment-winning.html)

[Prepare Where the Node Starts](https://ida.interchain.io/hands-on-exercise/2-ignite-cli-adv/6-payment-winning.html)

[Prepare and Connect to Other Nodes](https://ida.interchain.io/hands-on-exercise/2-ignite-cli-adv/6-payment-winning.html)

[Configure, Run, and Set Up a Service](https://ida.interchain.io/hands-on-exercise/2-ignite-cli-adv/6-payment-winning.html)

[Prepare and Do Migrations](https://ida.interchain.io/hands-on-exercise/2-ignite-cli-adv/6-payment-winning.html)

[Simulate Production in Docker](https://ida.interchain.io/hands-on-exercise/2-ignite-cli-adv/6-payment-winning.html)

[Tally Player Info After Production](https://ida.interchain.io/hands-on-exercise/2-ignite-cli-adv/6-payment-winning.html)

[Add a Leaderboard as a Module](https://ida.interchain.io/hands-on-exercise/2-ignite-cli-adv/6-payment-winning.html)

[Migrate the Leaderboard Module After Production](https://ida.interchain.io/hands-on-exercise/2-ignite-cli-adv/6-payment-winning.html)

[Simulate a Migration in Docker](https://ida.interchain.io/hands-on-exercise/2-ignite-cli-adv/6-payment-winning.html)

[Final Exam](https://ida.interchain.io/hands-on-exercise/2-ignite-cli-adv/6-payment-winning.html)

[](https://ida.interchain.io/hands-on-exercise/2-ignite-cli-adv/6-payment-winning.html)

[What's Next?](https://ida.interchain.io/hands-on-exercise/2-ignite-cli-adv/6-payment-winning.html)

[Continue Your Interchain Journey](https://ida.interchain.io/hands-on-exercise/2-ignite-cli-adv/6-payment-winning.html)

Docs Version Switcher

On this page

[Some initial thoughts](https://ida.interchain.io/hands-on-exercise/2-ignite-cli-adv/6-payment-winning.html#some-initial-thoughts)

[Code needs](https://ida.interchain.io/hands-on-exercise/2-ignite-cli-adv/6-payment-winning.html#code-needs)

[What is to be done](https://ida.interchain.io/hands-on-exercise/2-ignite-cli-adv/6-payment-winning.html#what-is-to-be-done)

[Declaring expectations](https://ida.interchain.io/hands-on-exercise/2-ignite-cli-adv/6-payment-winning.html#declaring-expectations)

[Obtaining the capability](https://ida.interchain.io/hands-on-exercise/2-ignite-cli-adv/6-payment-winning.html#obtaining-the-capability)

[Preparing expected errors](https://ida.interchain.io/hands-on-exercise/2-ignite-cli-adv/6-payment-winning.html#preparing-expected-errors)

[Money handling steps](https://ida.interchain.io/hands-on-exercise/2-ignite-cli-adv/6-payment-winning.html#money-handling-steps)

[Insert wager handling](https://ida.interchain.io/hands-on-exercise/2-ignite-cli-adv/6-payment-winning.html#insert-wager-handling)

[Unit tests](https://ida.interchain.io/hands-on-exercise/2-ignite-cli-adv/6-payment-winning.html#unit-tests)

[Prepare mocks](https://ida.interchain.io/hands-on-exercise/2-ignite-cli-adv/6-payment-winning.html#prepare-mocks)

[Make use of mocks](https://ida.interchain.io/hands-on-exercise/2-ignite-cli-adv/6-payment-winning.html#make-use-of-mocks)

[Adjust the unit tests](https://ida.interchain.io/hands-on-exercise/2-ignite-cli-adv/6-payment-winning.html#adjust-the-unit-tests)

[Wager handler unit tests](https://ida.interchain.io/hands-on-exercise/2-ignite-cli-adv/6-payment-winning.html#wager-handler-unit-tests)

[Add bank escrow unit tests](https://ida.interchain.io/hands-on-exercise/2-ignite-cli-adv/6-payment-winning.html#add-bank-escrow-unit-tests)

[Interact via the CLI](https://ida.interchain.io/hands-on-exercise/2-ignite-cli-adv/6-payment-winning.html#interact-via-the-cli)

[A game that expires](https://ida.interchain.io/hands-on-exercise/2-ignite-cli-adv/6-payment-winning.html#a-game-that-expires)

[A game played twice](https://ida.interchain.io/hands-on-exercise/2-ignite-cli-adv/6-payment-winning.html#a-game-played-twice)

[#Copy link](https://ida.interchain.io/hands-on-exercise/2-ignite-cli-adv/6-payment-winning.html#handle-wager-payments) **Handle Wager Payments**



Make sure you have everything you need before proceeding:

* You understand the concepts of [modules](https://ida.interchain.io/academy/2-cosmos-concepts/5-modules.html) and [keepers](https://ida.interchain.io/academy/2-cosmos-concepts/7-multistore-keepers.html).
* Go is installed.
* You have the checkers blockchain codebase up to the game wager. If not, follow the [previous steps](https://ida.interchain.io/hands-on-exercise/2-ignite-cli-adv/5-game-wager.html) or check out [the relevant version (opens new window)↗](https://github.com/cosmos/b9-checkers-academy-draft/tree/game-wager).



In this section, you will:

* Work with the bank module.
* Handle money.
* Use mocks.

In the [previous section](https://ida.interchain.io/hands-on-exercise/2-ignite-cli-adv/5-game-wager.html), you introduced a wager. On its own, having a Wager field is just a piece of information, it does not transfer tokens just by existing.

Transferring tokens is what this section is about.

[#Copy link](https://ida.interchain.io/hands-on-exercise/2-ignite-cli-adv/6-payment-winning.html#some-initial-thoughts) Some initial thoughts

When thinking about implementing a wager on games, ask:

* Is there any desirable atomicity of actions?
* At what junctures do you need to handle payments, refunds, and wins?
* Are there errors to report back?
* What event should you emit?

In the case of this example, you can consider that:

* Although a game creator can decide on a wager, it should only be the holder of the tokens that can decide when they are being taken from their balance.
* You might think of adding a new message type, one that indicates that a player puts its wager in escrow. On the other hand, you can leverage the existing messages and consider that when a player makes their first move, this expresses a willingness to participate, and therefore the tokens can be transferred at this juncture.
* For wins and losses, it is easy to imagine that the code handles the payout at the time a game is resolved.

[#Copy link](https://ida.interchain.io/hands-on-exercise/2-ignite-cli-adv/6-payment-winning.html#code-needs) Code needs

When it comes to your code:

* What Ignite CLI commands, if any, will assist you?
* How do you adjust what Ignite CLI created for you?
* Where do you make your changes?
* How would you unit-test these new elements?
* How would you use Ignite CLI to locally run a one-node blockchain and interact with it via the CLI to see what you get?

Here are some elements of response:

* Your module needs to call the bank to tell it to move tokens.
* Your module needs to be allowed by the bank to keep tokens in escrow.
* How would you test your module when it has such dependencies on the bank?

[#Copy link](https://ida.interchain.io/hands-on-exercise/2-ignite-cli-adv/6-payment-winning.html#what-is-to-be-done) What is to be done

A lot is to be done. In order you will:

* Make it possible for your checkers module to call certain functions of the bank to transfer tokens.
* Tell the bank to allow your checkers module to hold tokens in escrow.
* Create helper functions that encapsulate some knowledge about when and how to transfer tokens.
* Use these helper functions at the right places in your code.
* Update your unit tests and make use of mocks for that. You will create the mocks, create helper functions and use all that.

[#Copy link](https://ida.interchain.io/hands-on-exercise/2-ignite-cli-adv/6-payment-winning.html#declaring-expectations) Declaring expectations

On its own the Wager field does not make players pay the wager or receive rewards. You need to add handling actions that ask the bank module to perform the required token transfers. For that, your keeper needs to ask for a bank instance during setup.



The only way to have access to a capability with the object-capability model of the Cosmos SDK is to be given the reference to an instance which already has this capability.

Payment handling is implemented by having your keeper hold wagers **in escrow** while the game is being played. The bank module has functions to transfer tokens from any account to your module and vice-versa.

Alternatively, your keeper could burn tokens instead of keeping them in escrow and mint them again when paying out. However, this makes your blockchain's total supply *falsely* fluctuate. Additionally, this burning and minting may prove questionable when you later introduce IBC tokens.



Declare an interface that narrowly declares the functions from other modules that you expect for your module. The conventional file for these declarations is x/checkers/types/expected\_keepers.go.

The bank module has many capabilities, but all you need here are two functions. Your module already expects one function of the bank keeper: [SpendableCoins (opens new window)↗](https://github.com/cosmos/b9-checkers-academy-draft/blob/payment-winning/x/checkers/types/expected_keepers.go#L15-L18). Instead of expanding this interface, you add a new one and *redeclare* the extra functions you need like so:



Copy

type BankEscrowKeeper interface {

SendCoinsFromModuleToAccount(ctx sdk.Context, senderModule string, recipientAddr sdk.AccAddress, amt sdk.Coins) error

SendCoinsFromAccountToModule(ctx sdk.Context, senderAddr sdk.AccAddress, recipientModule string, amt sdk.Coins) error

}

x /

checkers /

types /

expected\_keepers.go

[View source→](https://github.com/cosmos/b9-checkers-academy-draft/blob/payment-winning/x/checkers/types/expected_keepers.go" \l "L20-L23" \t "_blank)

These two functions must exactly match the functions declared in the [bank's keeper.go file (opens new window)↗](https://github.com/cosmos/cosmos-sdk/blob/v0.45.4/x/bank/keeper/keeper.go#L35-L37). Copy the declarations directly from the bank's file. In Go, any object with these two functions is a BankEscrowKeeper.

[#Copy link](https://ida.interchain.io/hands-on-exercise/2-ignite-cli-adv/6-payment-winning.html#obtaining-the-capability) Obtaining the capability

With your requirements declared, it is time to make sure your keeper receives a reference to a bank keeper. First add a BankEscrowKeeper to your keeper in x/checkers/keeper/keeper.go:



Copy

type (

Keeper struct {

+ bank types.BankEscrowKeeper

...

}

)

x /

checkers /

keeper /

keeper.go

[View source→](https://github.com/cosmos/b9-checkers-academy-draft/blob/payment-winning/x/checkers/keeper/keeper.go" \l "L16" \t "_blank)

This BankEscrowKeeper is your newly declared narrow interface. Do not forget to adjust the constructor accordingly:



Copy

func NewKeeper(

+ bank types.BankEscrowKeeper,

...

) \*Keeper {

return &Keeper{

+ bank: bank,

...

}

}

x /

checkers /

keeper /

keeper.go

[View source→](https://github.com/cosmos/b9-checkers-academy-draft/blob/payment-winning/x/checkers/keeper/keeper.go" \l "L25-L38" \t "_blank)

Next, update where the constructor is called and pass a proper instance of BankKeeper. This happens in app/app.go:



Copy

app.CheckersKeeper = \*checkersmodulekeeper.NewKeeper(

+ app.BankKeeper,

...

)

app /

app.go

[View source→](https://github.com/cosmos/b9-checkers-academy-draft/blob/payment-winning/app/app.go" \l "L394" \t "_blank)

This app.BankKeeper is a full bank keeper that also conforms to your BankEscrowKeeper interface.

Finally, inform the app that your checkers module is going to hold balances in escrow by adding it to the **whitelist** of permitted modules:



Copy

maccPerms = map[string][]string{

...

+ checkersmoduletypes.ModuleName: nil,

}

app /

app.go

[View source→](https://github.com/cosmos/b9-checkers-academy-draft/blob/payment-winning/app/app.go" \l "L173" \t "_blank)

If you compare it to the other maccperms lines, the new line does not mention any authtypes.Minter or authtypes.Burner. Indeed nil is what you need to keep in escrow. For your information, the bank creates an *address* for your module's escrow account. When you have the full app, you can access it with:



Copy

import(

"github.com/alice/checkers/x/checkers/types"

)

checkersModuleAddress := app.AccountKeeper.GetModuleAddress(types.ModuleName)

[#Copy link](https://ida.interchain.io/hands-on-exercise/2-ignite-cli-adv/6-payment-winning.html#preparing-expected-errors) Preparing expected errors

There are several new error situations that you can enumerate with new variables:



Copy

var (

...

+ ErrBlackCannotPay = sdkerrors.Register(ModuleName, 1110, "black cannot pay the wager")

+ ErrRedCannotPay = sdkerrors.Register(ModuleName, 1111, "red cannot pay the wager")

+ ErrNothingToPay = sdkerrors.Register(ModuleName, 1112, "there is nothing to pay, should not have been called")

+ ErrCannotRefundWager = sdkerrors.Register(ModuleName, 1113, "cannot refund wager to: %s")

+ ErrCannotPayWinnings = sdkerrors.Register(ModuleName, 1114, "cannot pay winnings to winner: %s")

+ ErrNotInRefundState = sdkerrors.Register(ModuleName, 1115, "game is not in a state to refund, move count: %d")

)

x /

checkers /

types /

errors.go

[View source→](https://github.com/cosmos/b9-checkers-academy-draft/blob/payment-winning/x/checkers/types/errors.go" \l "L24-L29" \t "_blank)

[#Copy link](https://ida.interchain.io/hands-on-exercise/2-ignite-cli-adv/6-payment-winning.html#money-handling-steps) Money handling steps

With the bank now in your keeper, it is time to have your keeper handle the money. Keep this concern in its own file, as the functions are reused on play and forfeit.

Create the new file x/checkers/keeper/wager\_handler.go and add three functions to collect a wager, refund a wager, and pay winnings:



Copy

func (k \*Keeper) CollectWager(ctx sdk.Context, storedGame \*types.StoredGame) error

func (k \*Keeper) MustPayWinnings(ctx sdk.Context, storedGame \*types.StoredGame)

func (k \*Keeper) MustRefundWager(ctx sdk.Context, storedGame \*types.StoredGame)

x /

checkers /

keeper /

wager\_handler.go

[View source→](https://github.com/cosmos/b9-checkers-academy-draft/blob/payment-winning/x/checkers/keeper/wager_handler.go" \t "_blank)

The Must prefix in the function expresses the fact that the transaction either takes place or a panic is issued. Indeed, if a player cannot pay the wager, it is a user-side error and the user must be informed of a failed transaction. If the module cannot pay, it means the escrow account has failed. This latter error is much more serious: an invariant may have been violated and the whole application must be terminated.

Now set up collecting a wager, paying winnings, and refunding a wager:

1. **Collecting wagers** happens on a player's first move. Therefore, differentiate between players:



Copy

if storedGame.MoveCount == 0 {

// Black plays first

} else if storedGame.MoveCount == 1 {

// Red plays second

}

return nil

x /

checkers /

keeper /

wager\_handler.go

[View source→](https://github.com/cosmos/b9-checkers-academy-draft/blob/payment-winning/x/checkers/keeper/wager_handler.go" \l "L12-L33" \t "_blank)

When there are no moves, get the address for the black player:



Copy

black, err := storedGame.GetBlackAddress()

if err != nil {

panic(err.Error())

}

x /

checkers /

keeper /

wager\_handler.go

[View source→](https://github.com/cosmos/b9-checkers-academy-draft/blob/payment-winning/x/checkers/keeper/wager_handler.go" \l "L14-L17" \t "_blank)

Try to transfer into the escrow:



Copy

err = k.bank.SendCoinsFromAccountToModule(ctx, black, types.ModuleName, sdk.NewCoins(storedGame.GetWagerCoin()))

if err != nil {

return sdkerrors.Wrapf(err, types.ErrBlackCannotPay.Error())

}

x /

checkers /

keeper /

wager\_handler.go

[View source→](https://github.com/cosmos/b9-checkers-academy-draft/blob/payment-winning/x/checkers/keeper/wager_handler.go" \l "L18-L21" \t "_blank)

Then do the same for the [red player (opens new window)↗](https://github.com/cosmos/b9-checkers-academy-draft/blob/payment-winning/x/checkers/keeper/wager_handler.go#L24-L31) when there is a single move.

1. **Paying winnings** takes place when the game has a declared winner. First get the winner. "No winner" is **not** an acceptable situation in this MustPayWinnings. The caller of the function must ensure there is a winner:



Copy

winnerAddress, found, err := storedGame.GetWinnerAddress()

if err != nil {

panic(err.Error())

}

if !found {

panic(fmt.Sprintf(types.ErrCannotFindWinnerByColor.Error(), storedGame.Winner))

}

x /

checkers /

keeper /

wager\_handler.go

[View source→](https://github.com/cosmos/b9-checkers-academy-draft/blob/payment-winning/x/checkers/keeper/wager_handler.go" \l "L37-L43" \t "_blank)

Then calculate the winnings to pay:



Copy

winnings := storedGame.GetWagerCoin()

if storedGame.MoveCount == 0 {

panic(types.ErrNothingToPay.Error())

} else if 1 < storedGame.MoveCount {

winnings = winnings.Add(winnings)

}

x /

checkers /

keeper /

wager\_handler.go

[View source→](https://github.com/cosmos/b9-checkers-academy-draft/blob/payment-winning/x/checkers/keeper/wager_handler.go" \l "L44-L49" \t "_blank)

You double the wager only if the red player has also played and therefore both players have paid their wagers.

If you did this wrongly, you could end up in a situation where a game with a single move pays out as if both players had played. This would be a serious bug that an attacker could exploit to drain your module's escrow fund.

Then pay the winner:



Copy

err = k.bank.SendCoinsFromModuleToAccount(ctx, types.ModuleName, winnerAddress, sdk.NewCoins(winnings))

if err != nil {

panic(fmt.Sprintf(types.ErrCannotPayWinnings.Error(), err.Error()))

}

x /

checkers /

keeper /

wager\_handler.go

[View source→](https://github.com/cosmos/b9-checkers-academy-draft/blob/payment-winning/x/checkers/keeper/wager_handler.go" \l "L50-L53" \t "_blank)

1. Finally, **refunding wagers** takes place when the game has partially started, i.e. only one party has paid, or when the game ends in a draw. In this narrow case of MustRefundWager:



Copy

if storedGame.MoveCount == 1 {

// Refund

} else if storedGame.MoveCount == 0 {

// Do nothing

} else {

// TODO Implement a draw mechanism.

panic(fmt.Sprintf(types.ErrNotInRefundState.Error(), storedGame.MoveCount))

}

x /

checkers /

keeper /

wager\_handler.go

[View source→](https://github.com/cosmos/b9-checkers-academy-draft/blob/payment-winning/x/checkers/keeper/wager_handler.go" \l "L57-L72" \t "_blank)

Refund the black player when there has been a single move:



Copy

black, err := storedGame.GetBlackAddress()

if err != nil {

panic(err.Error())

}

err = k.bank.SendCoinsFromModuleToAccount(ctx, types.ModuleName, black, sdk.NewCoins(storedGame.GetWagerCoin()))

if err != nil {

panic(fmt.Sprintf(types.ErrCannotRefundWager.Error(), err.Error()))

}

x /

checkers /

keeper /

wager\_handler.go

[View source→](https://github.com/cosmos/b9-checkers-academy-draft/blob/payment-winning/x/checkers/keeper/wager_handler.go" \l "L59-L66" \t "_blank)

If the module cannot pay, then there is a panic as the escrow has failed.

You will notice that no special case is made when the wager is zero. This is a design choice here, and which way you choose to go is up to you. Not contacting the bank unnecessarily is cheaper in gas. On the other hand, why not outsource the zero check to the bank?

[#Copy link](https://ida.interchain.io/hands-on-exercise/2-ignite-cli-adv/6-payment-winning.html#insert-wager-handling) Insert wager handling

With the desired steps defined in the wager handling functions, it is time to invoke them at the right places in the message handlers.

1. When a player plays for the first time:



Copy

...

if !game.TurnIs(player) {

return nil, sdkerrors.Wrapf(types.ErrNotPlayerTurn, "%s", player)

}

+ err = k.Keeper.CollectWager(ctx, &storedGame)

+ if err != nil {

+ return nil, err

+ }

...

x /

checkers /

keeper /

msg\_server\_play\_move.go

[View source→](https://github.com/cosmos/b9-checkers-academy-draft/blob/payment-winning/x/checkers/keeper/msg_server_play_move.go" \l "L47-L50" \t "_blank)

1. When a player wins as a result of a move:



Copy

if storedGame.Winner == rules.PieceStrings[rules.NO\_PLAYER] {

...

} else {

...

k.Keeper.RemoveFromFifo(ctx, &storedGame, &systemInfo)

+ k.Keeper.MustPayWinnings(ctx, &storedGame)

}

x /

checkers /

keeper /

msg\_server\_play\_move.go

[View source→](https://github.com/cosmos/b9-checkers-academy-draft/blob/payment-winning/x/checkers/keeper/msg_server_play_move.go" \l "L79" \t "_blank)

1. When a game expires and there is a forfeit, make sure to only refund or pay full winnings when applicable. The logic needs to be adjusted:



Copy

if deadline.Before(ctx.BlockTime()) {

...

if storedGame.MoveCount <= 1 {

k.RemoveStoredGame(ctx, gameIndex)

+ if storedGame.MoveCount == 1 {

+ k.MustRefundWager(ctx, &storedGame)

+ }

} else {

...

+ k.MustPayWinnings(ctx, &storedGame)

storedGame.Board = ""

...

}

}

x /

checkers /

keeper /

end\_block\_server\_game.go

[View source→](https://github.com/cosmos/b9-checkers-academy-draft/blob/payment-winning/x/checkers/keeper/end_block_server_game.go" \l "L45-L56" \t "_blank)

[#Copy link](https://ida.interchain.io/hands-on-exercise/2-ignite-cli-adv/6-payment-winning.html#unit-tests) Unit tests

If you try running your existing tests you get a compilation error on the [test keeper builder (opens new window)↗](https://github.com/cosmos/b9-checkers-academy-draft/blob/payment-winning/testutil/keeper/checkers.go#L44-L49). Passing nil would not get you far with the tests and creating a full-fledged bank keeper would be a lot of work and not a unit test. See the next section on [integration tests](https://ida.interchain.io/hands-on-exercise/2-ignite-cli-adv/7-integration-tests.html) for that.

Instead, you create mocks and use them in unit tests, not only to get the existing tests to pass but also to verify that the bank is called as expected.

[#Copy link](https://ida.interchain.io/hands-on-exercise/2-ignite-cli-adv/6-payment-winning.html#prepare-mocks) Prepare mocks

It is better to create some [mocks (opens new window)↗](https://en.wikipedia.org/wiki/Mock_object). The Cosmos SDK does not offer mocks of its objects so you have to create your own. For that, the [gomock (opens new window)↗](https://github.com/golang/mock) library is a good resource. Install it:

**Local**

**Docker**



Copy

$ go install github.com/golang/mock/mockgen@v1.6.0

Copy

...

ENV NODE\_VERSION=18.x

ENV MOCKGEN\_VERSION=1.6.0

...

RUN apt-get install -y nodejs

# Install Mockgen

RUN go install github.com/golang/mock/mockgen@v${MOCKGEN\_VERSION}

...

Dockerfile-ubuntu

[View source](https://github.com/cosmos/b9-checkers-academy-draft/blob/payment-winning/Dockerfile-ubuntu" \l "L8-L33" \t "_blank)

Rebuild your Docker image.

With the library installed, you still need to do a one time creation of the mocks. Run:

**Local**

**Docker**



Copy

$ mockgen -source=x/checkers/types/expected\_keepers.go \

-package testutil \

-destination=x/checkers/testutil/expected\_keepers\_mocks.go

Copy

$ docker run --rm -it \

-v $(pwd):/checkers \

-w /checkers \

checkers\_i \

mockgen \

-source=x/checkers/types/expected\_keepers.go \

-package testutil \

-destination=x/checkers/testutil/expected\_keepers\_mocks.go

If your expected keepers change, you will have to run this command again. It can be a good idea to save the command for future reference. You may use a Makefile for that. Ensure you install the make tool for your computer. If you use Docker, add it to the packages and rebuild the image:



Copy

ENV PACKAGES curl gcc jq make

Dockerfile-ubuntu

[View source→](https://github.com/cosmos/b9-checkers-academy-draft/blob/payment-winning/Dockerfile-ubuntu" \l "L18" \t "_blank)

Create the Makefile:



Copy

mock-expected-keepers:

mockgen -source=x/checkers/types/expected\_keepers.go \

-package testutil \

-destination=x/checkers/testutil/expected\_keepers\_mocks.go

Makefile

[View source→](https://github.com/cosmos/b9-checkers-academy-draft/blob/payment-winning/Makefile" \l "L1-L4" \t "_blank)

At any time, you can rebuild the mocks with:

**Local**

**Docker**



Copy

$ make mock-expected-keepers

Copy

$ docker run --rm -it \

-v $(pwd):/checkers \

-w /checkers \

checkers\_i \

make mock-expected-keepers

You are going to set the expectations on this BankEscrowKeeper mock many times, including when you do not care about the result. So instead of setting the verbose expectations in every test, it is in your interest to create helper functions that will make setting up the expectations more succinct and therefore readable, which is always a benefit when unit testing. Create a new bank\_escrow\_helpers.go file with:

* A function to *unset* expectations (ie where anything can go). This is useful when your test does not check things around the escrow:



Copy

func (escrow \*MockBankEscrowKeeper) ExpectAny(context context.Context) {

escrow.EXPECT().SendCoinsFromAccountToModule(sdk.UnwrapSDKContext(context), gomock.Any(), gomock.Any(), gomock.Any()).AnyTimes()

escrow.EXPECT().SendCoinsFromModuleToAccount(sdk.UnwrapSDKContext(context), gomock.Any(), gomock.Any(), gomock.Any()).AnyTimes()

}

x /

checkers /

testutil /

bank\_escrow\_helpers.go

[View source→](https://github.com/cosmos/b9-checkers-academy-draft/blob/payment-winning/x/checkers/testutil/bank_escrow_helpers.go" \l "L11-L14" \t "_blank)

* A function to quickly create the right type and value of coins to use for expectations:



Copy

func coinsOf(amount uint64) sdk.Coins {

return sdk.Coins{

sdk.Coin{

Denom: sdk.DefaultBondDenom,

Amount: sdk.NewInt(int64(amount)),

},

}

}

x /

checkers /

testutil /

bank\_escrow\_helpers.go

[View source→](https://github.com/cosmos/b9-checkers-academy-draft/blob/payment-winning/x/checkers/testutil/bank_escrow_helpers.go" \l "L16-L23" \t "_blank)

* A function to define an expectation of payment from a player:



Copy

func (escrow \*MockBankEscrowKeeper) ExpectPay(context context.Context, who string, amount uint64) \*gomock.Call {

whoAddr, err := sdk.AccAddressFromBech32(who)

if err != nil {

panic(err)

}

return escrow.EXPECT().SendCoinsFromAccountToModule(sdk.UnwrapSDKContext(context), whoAddr, types.ModuleName, coinsOf(amount))

}

x /

checkers /

testutil /

bank\_escrow\_helpers.go

[View source→](https://github.com/cosmos/b9-checkers-academy-draft/blob/payment-winning/x/checkers/testutil/bank_escrow_helpers.go" \l "L25-L31" \t "_blank)

* A function to define an expectation of payment from the module:



Copy

func (escrow \*MockBankEscrowKeeper) ExpectRefund(context context.Context, who string, amount uint64) \*gomock.Call {

whoAddr, err := sdk.AccAddressFromBech32(who)

if err != nil {

panic(err)

}

return escrow.EXPECT().SendCoinsFromModuleToAccount(sdk.UnwrapSDKContext(context), types.ModuleName, whoAddr, coinsOf(amount))

}

x /

checkers /

testutil /

bank\_escrow\_helpers.go

[View source→](https://github.com/cosmos/b9-checkers-academy-draft/blob/payment-winning/x/checkers/testutil/bank_escrow_helpers.go" \l "L33-L39" \t "_blank)

Note that the two main expectation functions return \*gomock.Call. This is so that you can continue adding expectations such as:

* Instructing the number of times the call is expected to be made: for instance .Times(1).
* Instructing the order of the expected calls: for instance .After(<the previous call>).

[#Copy link](https://ida.interchain.io/hands-on-exercise/2-ignite-cli-adv/6-payment-winning.html#make-use-of-mocks) Make use of mocks

With the helpers in place, you can add a new function similar to CheckersKeeper(t testing.TB) but which uses mocks. Keep the original function, which passes a nil for bank:



Copy

func CheckersKeeper(t testing.TB) (\*keeper.Keeper, sdk.Context) {

+ return CheckersKeeperWithMocks(t, nil)

+ }

+ func CheckersKeeperWithMocks(t testing.TB, bank \*testutil.MockBankEscrowKeeper) (\*keeper.Keeper, sdk.Context) {

...

k := keeper.NewKeeper(

+ bank,

...

)

...

}

testutil /

keeper /

checkers.go

[View source→](https://github.com/cosmos/b9-checkers-academy-draft/blob/payment-winning/testutil/keeper/checkers.go" \l "L21-L58" \t "_blank)

The CheckersKeeperWithMocks function takes the mock in its arguments for more versatility. The CheckersKeeper remains for the tests that never call the escrow.

Now adjust the small functions that set up the keeper before each test. You do not need to change them for the *create* tests, because they never call the bank. You have to do it for *play* and *forfeit*.

For *play*:



Copy

- func setupMsgServerWithOneGameForPlayMove(t testing.TB) (types.MsgServer, keeper.Keeper, context.Context) {

- k, ctx := keepertest.CheckersKeeper(t)

+ func setupMsgServerWithOneGameForPlayMove(t testing.TB) (types.MsgServer, keeper.Keeper, context.Context,

+ \*gomock.Controller, \*testutil.MockBankEscrowKeeper) {

+ ctrl := gomock.NewController(t)

+ bankMock := testutil.NewMockBankEscrowKeeper(ctrl)

+ k, ctx := keepertest.CheckersKeeperWithMocks(t, bankMock)

checkers.InitGenesis(ctx, \*k, \*types.DefaultGenesis())

server := keeper.NewMsgServerImpl(\*k)

context := sdk.WrapSDKContext(ctx)

server.CreateGame(context, &types.MsgCreateGame{

Creator: alice,

Black: bob,

Red: carol,

Wager: 45,

})

- return server, \*k, context

+ return server, \*k, context, ctrl, bankMock

}

x /

checkers /

keeper /

msg\_server\_play\_move\_test.go

[View source→](https://github.com/cosmos/b9-checkers-academy-draft/blob/payment-winning/x/checkers/keeper/msg_server_play_move_test.go" \l "L17-L32" \t "_blank)

This function creates the mock and returns two new objects:

* The mock controller, so that the .Finish() method can be called within the test itself. This is the function that will verify the call expectations placed on the mocks.
* The mocked bank escrow. This is the instance on which you place the call expectations.

Both objects will be used from the tests proper.

Do the same for *forfeit* if their unit tests do not use the above setupMsgServerWithOneGameForPlayMove.

[#Copy link](https://ida.interchain.io/hands-on-exercise/2-ignite-cli-adv/6-payment-winning.html#adjust-the-unit-tests) Adjust the unit tests

With these changes, you need to adjust many unit tests for *play* and *forfeit*. Often you may only want to make the tests pass again without checking any meaningful bank call expectations. There are different situations:

1. The mocked bank is not called. Therefore, you do not add any expectation and still call the controller:



Copy

func TestForfeitUnplayed(t \*testing.T) {

- \_, keeper, context := setupMsgServerWithOneGameForPlayMove(t)

+ \_, keeper, context, ctrl, \_ := setupMsgServerWithOneGameForPlayMove(t)

ctx := sdk.UnwrapSDKContext(context)

+ defer ctrl.Finish()

...

}

x /

checkers /

keeper /

end\_block\_server\_game\_test.go

[View source→](https://github.com/cosmos/b9-checkers-academy-draft/blob/payment-winning/x/checkers/keeper/end_block_server_game_test.go" \l "L13-L15" \t "_blank)

When you expect the mocked bank **not** to be called, it is important that you do not put any expectations on it. This means the test will fail if it is called by mistake.

1. The mocked bank is called, but you do not care about how it is called:



Copy

- msgServer, \_, context := setupMsgServerWithOneGameForRejectGame(t)

+ msgServer, \_, context, ctrl, escrow := setupMsgServerWithOneGameForRejectGame(t)

+ defer ctrl.Finish()

+ escrow.ExpectAny(context)

Here you do not want to be distracted by escrow concerns.

1. The mocked bank is called, and you want to add call expectations:



Copy

func TestForfeitPlayedOnce(t \*testing.T) {

- msgServer, keeper, context := setupMsgServerWithOneGameForPlayMove(t)

+ msgServer, keeper, context, ctrl, escrow := setupMsgServerWithOneGameForPlayMove(t)

ctx := sdk.UnwrapSDKContext(context)

+ defer ctrl.Finish()

+ pay := escrow.ExpectPay(context, bob, 45).Times(1)

+ escrow.ExpectRefund(context, bob, 45).Times(1).After(pay)

...

}

x /

checkers /

keeper /

end\_block\_server\_game\_test.go

[View source→](https://github.com/cosmos/b9-checkers-academy-draft/blob/payment-winning/x/checkers/keeper/end_block_server_game_test.go" \l "L139-L143" \t "_blank)

Go ahead and make the many necessary changes as you see fit.

[#Copy link](https://ida.interchain.io/hands-on-exercise/2-ignite-cli-adv/6-payment-winning.html#wager-handler-unit-tests) Wager handler unit tests

After these adjustments, it is a good idea to add unit tests directly on the wager handling functions of the keeper. Create a new wager\_handler\_test.go file. In it:

1. Add a setup helper function that does not create any message server:



Copy

func setupKeeperForWagerHandler(t testing.TB) (keeper.Keeper, context.Context,

\*gomock.Controller, \*testutil.MockBankEscrowKeeper) {

ctrl := gomock.NewController(t)

bankMock := testutil.NewMockBankEscrowKeeper(ctrl)

k, ctx := keepertest.CheckersKeeperWithMocks(t, bankMock)

checkers.InitGenesis(ctx, \*k, \*types.DefaultGenesis())

context := sdk.WrapSDKContext(ctx)

return \*k, context, ctrl, bankMock

}

x /

checkers /

keeper /

wager\_handler\_test.go

[View source→](https://github.com/cosmos/b9-checkers-academy-draft/blob/payment-winning/x/checkers/keeper/wager_handler_test.go" \l "L18-L26" \t "_blank)

1. Add tests on the CollectWager function. For instance, when the game is malformed:



Copy

func TestWagerHandlerCollectWrongNoBlack(t \*testing.T) {

keeper, context, ctrl, \_ := setupKeeperForWagerHandler(t)

ctx := sdk.UnwrapSDKContext(context)

defer ctrl.Finish()

defer func() {

r := recover()

require.NotNil(t, r, "The code did not panic")

require.Equal(t, "black address is invalid: : empty address string is not allowed", r)

}()

keeper.CollectWager(ctx, &types.StoredGame{

MoveCount: 0,

})

}

x /

checkers /

keeper /

wager\_handler\_test.go

[View source→](https://github.com/cosmos/b9-checkers-academy-draft/blob/payment-winning/x/checkers/keeper/wager_handler_test.go" \l "L28-L40" \t "_blank)

Or when the black player failed to escrow the wager:



Copy

func TestWagerHandlerCollectFailedNoMove(t \*testing.T) {

keeper, context, ctrl, escrow := setupKeeperForWagerHandler(t)

ctx := sdk.UnwrapSDKContext(context)

defer ctrl.Finish()

black, \_ := sdk.AccAddressFromBech32(alice)

escrow.EXPECT().

SendCoinsFromAccountToModule(ctx, black, types.ModuleName, gomock.Any()).

Return(errors.New("oops"))

err := keeper.CollectWager(ctx, &types.StoredGame{

Black: alice,

MoveCount: 0,

Wager: 45,

})

require.NotNil(t, err)

require.EqualError(t, err, "black cannot pay the wager: oops")

}

x /

checkers /

keeper /

wager\_handler\_test.go

[View source→](https://github.com/cosmos/b9-checkers-academy-draft/blob/payment-winning/x/checkers/keeper/wager_handler_test.go" \l "L42-L57" \t "_blank)

Or when the collection of a wager works:



Copy

func TestWagerHandlerCollectNoMove(t \*testing.T) {

keeper, context, ctrl, escrow := setupKeeperForWagerHandler(t)

ctx := sdk.UnwrapSDKContext(context)

defer ctrl.Finish()

escrow.ExpectPay(context, alice, 45)

err := keeper.CollectWager(ctx, &types.StoredGame{

Black: alice,

MoveCount: 0,

Wager: 45,

})

require.Nil(t, err)

}

x /

checkers /

keeper /

wager\_handler\_test.go

[View source→](https://github.com/cosmos/b9-checkers-academy-draft/blob/payment-winning/x/checkers/keeper/wager_handler_test.go" \l "L90-L101" \t "_blank)

1. Add similar tests to the payment of winnings from the escrow. When it fails:



Copy

func TestWagerHandlerPayWrongEscrowFailed(t \*testing.T) {

keeper, context, ctrl, escrow := setupKeeperForWagerHandler(t)

ctx := sdk.UnwrapSDKContext(context)

defer ctrl.Finish()

black, \_ := sdk.AccAddressFromBech32(alice)

escrow.EXPECT().

SendCoinsFromModuleToAccount(ctx, types.ModuleName, black, gomock.Any()).

Times(1).

Return(errors.New("oops"))

defer func() {

r := recover()

require.NotNil(t, r, "The code did not panic")

require.Equal(t, r, "cannot pay winnings to winner: oops")

}()

keeper.MustPayWinnings(ctx, &types.StoredGame{

Black: alice,

Red: bob,

Winner: "b",

MoveCount: 1,

Wager: 45,

})

}

x /

checkers /

keeper /

wager\_handler\_test.go

[View source→](https://github.com/cosmos/b9-checkers-academy-draft/blob/payment-winning/x/checkers/keeper/wager_handler_test.go" \l "L163-L184" \t "_blank)

Or when it works:



Copy

func TestWagerHandlerPayEscrowCalledTwoMoves(t \*testing.T) {

keeper, context, ctrl, escrow := setupKeeperForWagerHandler(t)

ctx := sdk.UnwrapSDKContext(context)

defer ctrl.Finish()

escrow.ExpectRefund(context, alice, 90)

keeper.MustPayWinnings(ctx, &types.StoredGame{

Black: alice,

Red: bob,

Winner: "b",

MoveCount: 2,

Wager: 45,

})

}

x /

checkers /

keeper /

wager\_handler\_test.go

[View source→](https://github.com/cosmos/b9-checkers-academy-draft/blob/payment-winning/x/checkers/keeper/wager_handler_test.go" \l "L200-L212" \t "_blank)

1. You will also need a test for [refund (opens new window)↗](https://github.com/cosmos/b9-checkers-academy-draft/blob/payment-winning/x/checkers/keeper/wager_handler_test.go#L251-L270) situations.

[#Copy link](https://ida.interchain.io/hands-on-exercise/2-ignite-cli-adv/6-payment-winning.html#add-bank-escrow-unit-tests) Add bank escrow unit tests

Now that the wager handling has been convincingly tested, you want to confirm that its functions are called at the right junctures. Add dedicated tests with message servers that confirm how the bank is called. Add them in existing files, for instance:



Copy

func TestPlayMoveUpToWinnerCalledBank(t \*testing.T) {

msgServer, \_, context, ctrl, escrow := setupMsgServerWithOneGameForPlayMove(t)

defer ctrl.Finish()

payBob := escrow.ExpectPay(context, bob, 45).Times(1)

payCarol := escrow.ExpectPay(context, carol, 45).Times(1).After(payBob)

escrow.ExpectRefund(context, bob, 90).Times(1).After(payCarol)

playAllMoves(t, msgServer, context, "1", testutil.Game1Moves)

}

x /

checkers /

keeper /

msg\_server\_play\_move\_winner\_test.go

[View source→](https://github.com/cosmos/b9-checkers-academy-draft/blob/payment-winning/x/checkers/keeper/msg_server_play_move_winner_test.go" \l "L57-L65" \t "_blank)

After doing all that, confirm that your tests run.

**Local**

**Docker**



Copy

$ go test github.com/alice/checkers/x/checkers/keeper

Copy

$ docker run --rm -it \

-v $(pwd):/checkers \

-w /checkers \

checkers\_i \

go test github.com/alice/checkers/x/checkers/keeper

You have confirmed that your code behaves correctly given the expectations you have placed on the bank keeper.

[#Copy link](https://ida.interchain.io/hands-on-exercise/2-ignite-cli-adv/6-payment-winning.html#interact-via-the-cli) Interact via the CLI

With the tests done, see what happens at the command-line and see if the bank keeper behaves as you expected.

Keep the game expiry at 5 minutes to be able to test a forfeit, as done in a [previous section](https://ida.interchain.io/hands-on-exercise/2-ignite-cli-adv/4-game-forfeit.html). Now, you need to check balances after relevant steps to test that wagers are being withheld and paid.

How much do Alice and Bob have to start with?

**Local**

**Docker**



Copy

$ checkersd query bank balances $alice

$ checkersd query bank balances $bob

Copy

$ docker exec -it checkers \

checkersd query bank balances $alice

$ docker exec -it checkers \

checkersd query bank balances $bob

This prints:



Copy

balances:

- amount: "100000000"

denom: stake

- amount: "20000"

denom: token

pagination:

next\_key: null

total: "0"

balances:

- amount: "100000000"

denom: stake

- amount: "10000"

denom: token

pagination:

next\_key: null

total: "0"

[#Copy link](https://ida.interchain.io/hands-on-exercise/2-ignite-cli-adv/6-payment-winning.html#a-game-that-expires) A game that expires

Create a game on which the wager will be refunded because the player playing red did not join:

**Local**

**Docker**



Copy

$ checkersd tx checkers create-game $alice $bob 1000000 --from $alice

Copy

$ docker exec -it checkers \

checkersd tx checkers create-game $alice $bob 1000000 --from $alice

Confirm that the balances of both Alice and Bob are unchanged - as they have not played yet.



In this example, Alice paid no gas fees, other than the transaction costs, to create a game. The gas price is likely 0 here anyway. This is fixed in the [next section](https://ida.interchain.io/hands-on-exercise/2-ignite-cli-adv/8-gas-meter.html).

Have Alice play:

**Local**

**Docker**



Copy

$ checkersd tx checkers play-move 1 1 2 2 3 --from $alice

Copy

$ docker exec -it checkers \

checkersd tx checkers play-move 1 1 2 2 3 --from $alice

Confirm that Alice has paid her wager:

**Local**

**Docker**



Copy

$ checkersd query bank balances $alice

Copy

$ docker exec -it checkers \

checkersd query bank balances $alice

This prints:



Copy

balances:

- amount: "99000000" # <- 1,000,000 fewer

denom: stake

- amount: "20000"

denom: token

pagination:

next\_key: null

total: "0"

Wait 5 minutes for the game to expire and check again:

**Local**

**Docker**



Copy

$ checkersd query bank balances $alice

Copy

$ docker exec -it checkers \

checkersd query bank balances $alice

This prints:



Copy

balances:

- amount: "100000000" # <- 1,000,000 are back

denom: stake

- amount: "20000"

denom: token

pagination:

next\_key: null

total: "0"

[#Copy link](https://ida.interchain.io/hands-on-exercise/2-ignite-cli-adv/6-payment-winning.html#a-game-played-twice) A game played twice

Now create a game in which both players only play once each, i.e. where the player playing black forfeits:

**Local**

**Docker**



Copy

$ checkersd tx checkers create-game $alice $bob 1000000 --from $alice

$ checkersd tx checkers play-move 2 1 2 2 3 --from $alice

$ checkersd tx checkers play-move 2 0 5 1 4 --from $bob

Copy

$ docker exec -it checkers \

checkersd tx checkers create-game $alice $bob 1000000 --from $alice

$ docker exec -it checkers \

checkersd tx checkers play-move 2 1 2 2 3 --from $alice

$ docker exec -it checkers \

checkersd tx checkers play-move 2 0 5 1 4 --from $bob

Confirm that both Alice and Bob paid their wagers. Wait 5 minutes for the game to expire and check again:

**Local**

**Docker**



Copy

$ checkersd query bank balances $alice

$ checkersd query bank balances $bob

Copy

$ docker exec -it checkers \

checkersd query bank balances $alice

$ docker exec -it checkers \

checkersd query bank balances $bob

This shows:



Copy

balances:

- amount: "99000000" # <- her 1,000,000 are gone for good

denom: stake

...

balances:

- amount: "101000000" # <- 1,000,000 more than at the beginning

denom: stake

...

This is correct: Bob was the winner by forfeit.

Similarly, you can test that Alice gets her wager back when Alice creates a game, Alice plays, and then Bob lets it expire.

It would be difficult to test by CLI when there is a winner after a full game. That would be better tested with a GUI, or by using integration tests as is done in the [next section](https://ida.interchain.io/hands-on-exercise/2-ignite-cli-adv/7-integration-tests.html).

synopsis

To summarize, this section has explored:

* How to work with the Bank module and handle players making wagers on games, now that the application supports live games playing to completion (with the winner claiming both wagers) or expiring through inactivity (with the inactive player forfeiting their wager as if losing), and no possibility of withheld value being stranded in inactive games.
* How to add handling actions that ask the bank module to perform the token transfers required by the wager, and where to invoke them in the message handlers.
* How to create a new wager-handling file with functions to collect a wager, refund a wager, and pay winnings, in which must prefixes indicate either a user-side error (leading to a failed transaction) or a failure of the application's escrow account (requiring the whole application be terminated).
* How to interact with the CLI to check account balances to test that wagers are being withheld and paid.

previous

[](https://ida.interchain.io/hands-on-exercise/2-ignite-cli-adv/5-game-wager.html)

**[Let Players Set a Wager](https://ida.interchain.io/hands-on-exercise/2-ignite-cli-adv/5-game-wager.html)**

up next

**[Integration tests](https://ida.interchain.io/hands-on-exercise/2-ignite-cli-adv/7-integration-tests.html)**

[[](https://ida.interchain.io/hands-on-exercise/2-ignite-cli-adv/7-integration-tests.html)](https://ida.interchain.io/hands-on-exercise/2-ignite-cli-adv/7-integration-tests.html)

Rate this Page

icon smile

icon meh

icon frown

Would you like to add a message?

Submit

Thank you for your Feedback!

[](https://ida.interchain.io/ida-course/discord-info.html)

On this page

[Some initial thoughts](https://ida.interchain.io/hands-on-exercise/2-ignite-cli-adv/6-payment-winning.html#some-initial-thoughts)

[Code needs](https://ida.interchain.io/hands-on-exercise/2-ignite-cli-adv/6-payment-winning.html#code-needs)

[What is to be done](https://ida.interchain.io/hands-on-exercise/2-ignite-cli-adv/6-payment-winning.html#what-is-to-be-done)

[Declaring expectations](https://ida.interchain.io/hands-on-exercise/2-ignite-cli-adv/6-payment-winning.html#declaring-expectations)

[Obtaining the capability](https://ida.interchain.io/hands-on-exercise/2-ignite-cli-adv/6-payment-winning.html#obtaining-the-capability)

[Preparing expected errors](https://ida.interchain.io/hands-on-exercise/2-ignite-cli-adv/6-payment-winning.html#preparing-expected-errors)

[Money handling steps](https://ida.interchain.io/hands-on-exercise/2-ignite-cli-adv/6-payment-winning.html#money-handling-steps)

[Insert wager handling](https://ida.interchain.io/hands-on-exercise/2-ignite-cli-adv/6-payment-winning.html#insert-wager-handling)

[Unit tests](https://ida.interchain.io/hands-on-exercise/2-ignite-cli-adv/6-payment-winning.html#unit-tests)

[Prepare mocks](https://ida.interchain.io/hands-on-exercise/2-ignite-cli-adv/6-payment-winning.html#prepare-mocks)

[Make use of mocks](https://ida.interchain.io/hands-on-exercise/2-ignite-cli-adv/6-payment-winning.html#make-use-of-mocks)

[Adjust the unit tests](https://ida.interchain.io/hands-on-exercise/2-ignite-cli-adv/6-payment-winning.html#adjust-the-unit-tests)

[Wager handler unit tests](https://ida.interchain.io/hands-on-exercise/2-ignite-cli-adv/6-payment-winning.html#wager-handler-unit-tests)

[Add bank escrow unit tests](https://ida.interchain.io/hands-on-exercise/2-ignite-cli-adv/6-payment-winning.html#add-bank-escrow-unit-tests)

[Interact via the CLI](https://ida.interchain.io/hands-on-exercise/2-ignite-cli-adv/6-payment-winning.html#interact-via-the-cli)

[A game that expires](https://ida.interchain.io/hands-on-exercise/2-ignite-cli-adv/6-payment-winning.html#a-game-that-expires)

[A game played twice](https://ida.interchain.io/hands-on-exercise/2-ignite-cli-adv/6-payment-winning.html#a-game-played-twice)

#### **Get Cosmos updates**

Unsubscribe at any time. [Privacy Policy↗](https://v1.cosmos.network/privacy)

     Next

Documentation

[Cosmos SDK](https://docs.cosmos.network/)[Cosmos Hub](https://hub.cosmos.network/)[CometBFT](https://docs.cometbft.com/)[IBC Protocol](https://ibc.cosmos.network/)

Community

[Interchain blog](https://blog.cosmos.network/)[Forum](https://forum.cosmos.network/)[Discord](https://discord.gg/cosmosnetwork)

Contributing

[Source code on GitHub](https://github.com/cosmos/sdk-tutorials)

[](https://ida.interchain.io/)

[Interchain Developer Academy](https://ida.interchain.io/)

**[](https://blog.cosmos.network/)[](https://twitter.com/cosmos)[](https://discord.gg/cosmosnetwork)[](https://www.linkedin.com/company/interchain-foundation/about/)[](https://reddit.com/r/cosmosnetwork)[](https://t.me/cosmosproject)[](https://www.youtube.com/c/CosmosProject)**



Dark mode

† This website is maintained by the Interchain Foundation (ICF). The contents and opinions of this website are those of the ICF. The ICF provides links to cryptocurrency exchanges as a service to the public. The ICF does not warrant that the information provided by these websites is correct, complete, and up-to-date. The ICF is not responsible for their content and expressly rejects any liability for damages of any kind resulting from the use, reference to, or reliance on any information contained within these websites.

Cosmos is a registered trademark of the [Interchain Foundation.](https://interchain.io/)[Privacy](https://v1.cosmos.network/privacy)

Original text

Rate this translation

Your feedback will be used to help improve Google Translate