Introduction to Tokens

FinTech

Lesson 21.1

Class Objectives

By the end of this lesson, you will be able to:

- Explain what a blockchain token is, what gives it value, and how it can be used.
- Distinguish the general difference between a coin and a token in the context of blockchain.
- Build a Solidity smart contract that creates a simple token.
- Use the mapping data structure to associate customer addresses with their individual information.
- Import third-party OpenZeppelin libraries from GitHub into a Solidity smart contract.
- Use the SafeMath library to perform secure math operations.
- Explain the roles that inheritance and composition play in object-oriented programming (OOP).

Before we move on to new and advanced concepts in Solidity, let’s review some of the basics that we already know.

What’s Solidity?

What Is Solidity?

01

A high-level object-oriented programing language

02

The language used to write smart contracts on the Ethereum blockchain

03

A strictly typed language

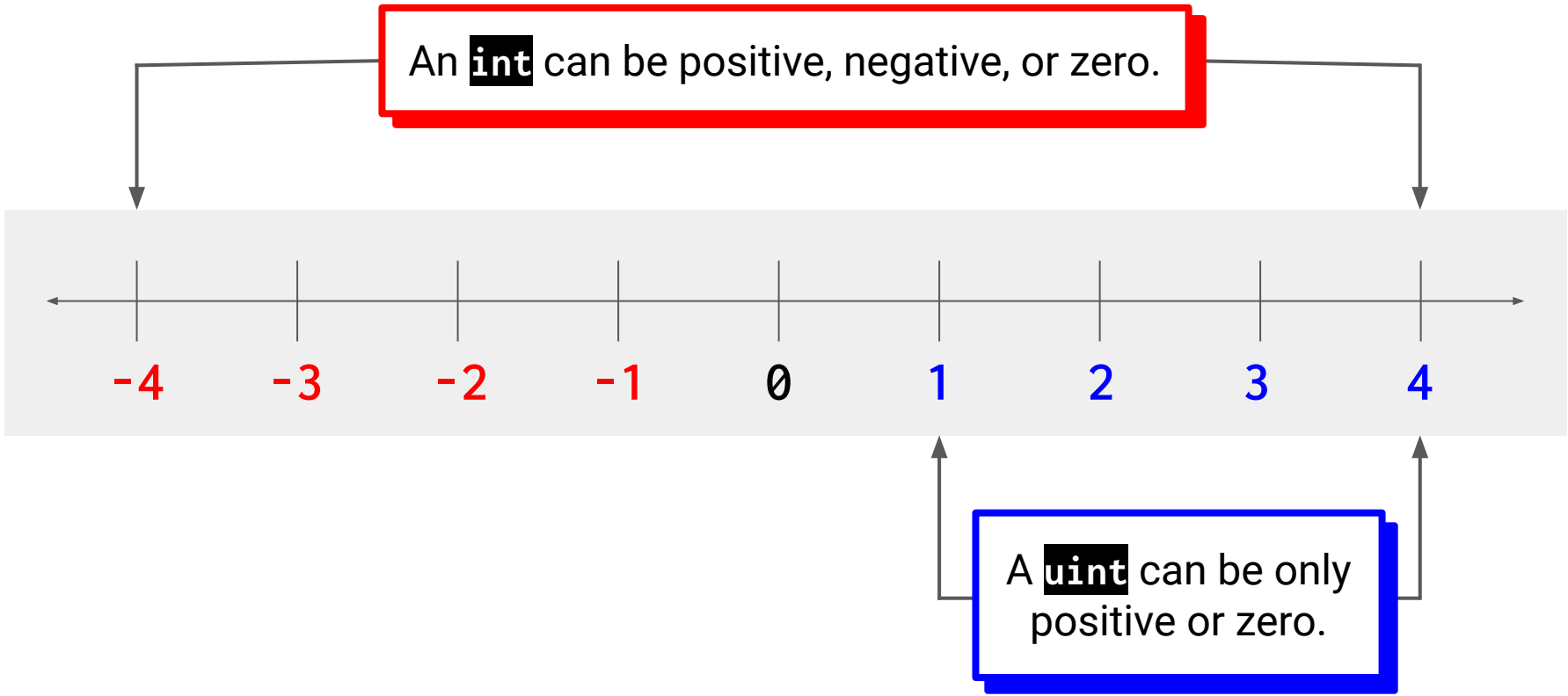
What's a `uint`?

Unsigned Integer

A `uint` is an unsigned integer that accepts only positive numbers ranging from 0 to (2^256)−1.

```
uint: 0 to (2^256)-1
```

What's the difference between an `int` and a `uint`?


An int can be positive, negative, or zero.
-4
-3
-2
-1
0
1
2
3
4
A uint can be only positive or zero.

Why is Solidity so strict with its typing?

Solidity Typing

Solidity is strict with its typing because:

It allows for better error handling in code.

Contracts should not leave room for ambiguity.

Being upfront about data types and the size to store them results in less computational overhead/gas costs.

What's a `payable` address, and how does it differ from a regular address?

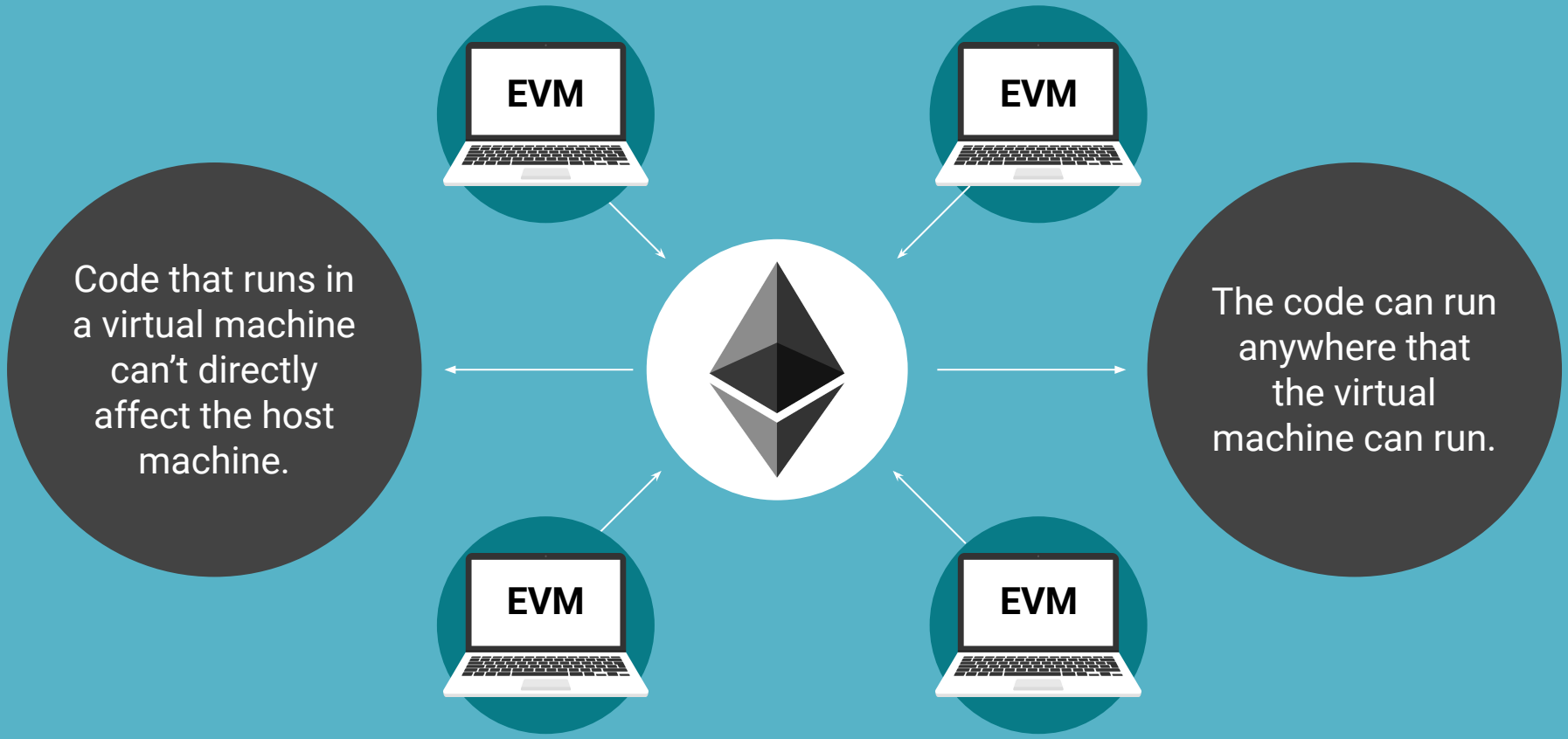
Payable Address

A `payable` address resembles a regular address, except that it allows calling the `transfer` function to send ether to it.

```
address.transfer(amountEther)
```

What's a potential benefit of running code in a virtual machine?

Ethereum Virtual Machine



Introduction to Tokenomics

In the traditional financial market, when a company wants to raise capital, they can sell shares of the company to the public through an **initial public offering (IPO)**.

Blockchain companies have another option available to them:

Raising funds directly on the blockchain through a similar process, called an **initial coin offering (ICO)**.

Introduction to Tokenomics

When a company holds an ICO, they use blockchain technology to generate and sell either:

Blockchain coins

Cyptocurrencies that are native to their blockchains, such as bitcoin

Digital tokens

Represent any asset, utility, service, or currency that has value to the potential buyers

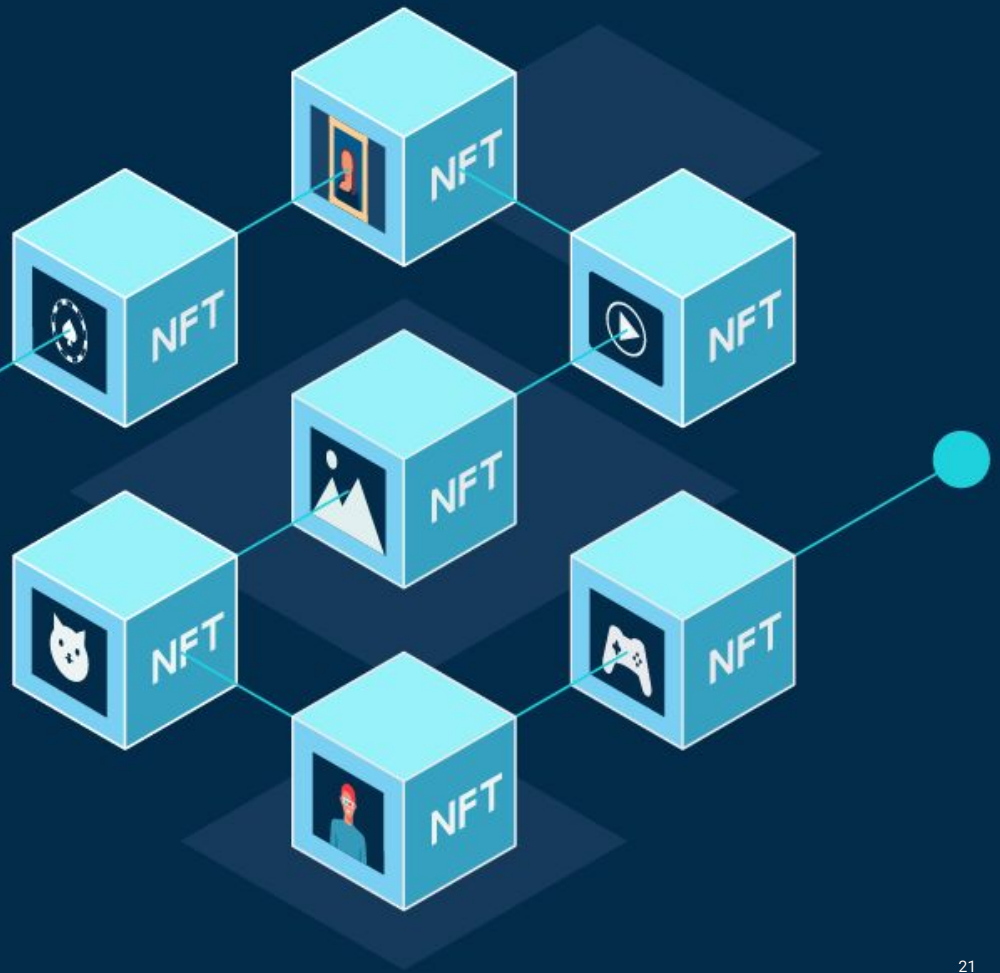
Introduction to Tokenomics

- Tokens might represent equity in the company (like in a traditional IPO) or a promise of future payment.
- They might also represent a utility or benefit that relates to the company's blockchain application or service.
- These benefits include extra privileges on the system or exclusive or early access.
- In a gaming system, for example, a token might represent a special or unique item in the game.

Regardless of the asset that ICO tokens represent, the blockchain uses a particular type of smart contract to encode the details of the token's value.

While blockchain tokens are often produced through an ICO, it's possible to produce them without undertaking an ICO.

This lesson will teach you how to code blockchain token smart contracts without an ICO.



Tokenomics

Tokenomics, or the economics of tokens, refers to how blockchain tokens get conceptualized, produced, valued, distributed, traded, and used.

Introduction to Tokenomics

A blockchain token represents an asset or utility on a blockchain platform. Essentially, it’s a symbol of value.

=

Introduction to Tokenomics

Imagine an arcade. When entering the arcade, customers might exchange cash for arcade tokens that are worth 50 cents each.

- Within the ecosystem of the arcade, an arcade token represents a customer's ownership of 50 cents.
- That customer can exchange the arcade token for a certain amount of time playing a game.
- Or, they can transfer their ownership of the 50 cents to someone else by giving the arcade token to another customer.

Introduction to Tokenomics

Blockchain tokens function much the same way.

Blockchain tokens get rendered digitally rather than as physical objects.

On a blockchain, an asset can be tokenized, or represented as a token.

Introduction to Tokenomics

Commodities, like gold and silver, and currencies, like the US dollar, can all be **Virtually anything that holds value can be represented as a token on a blockchain.**.

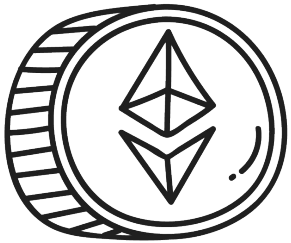
Tokens and Smart Contracts

Tokens and Smart Contracts

We create tokens on the Ethereum blockchain by using smart contracts.

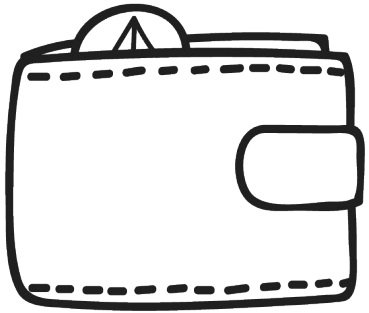
Token

The token itself is the symbol of value.



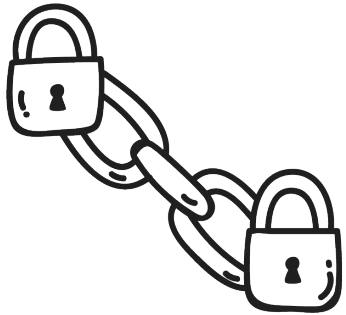
Wallet

The ownership of a token gets represented in the participant's wallet.



Contract

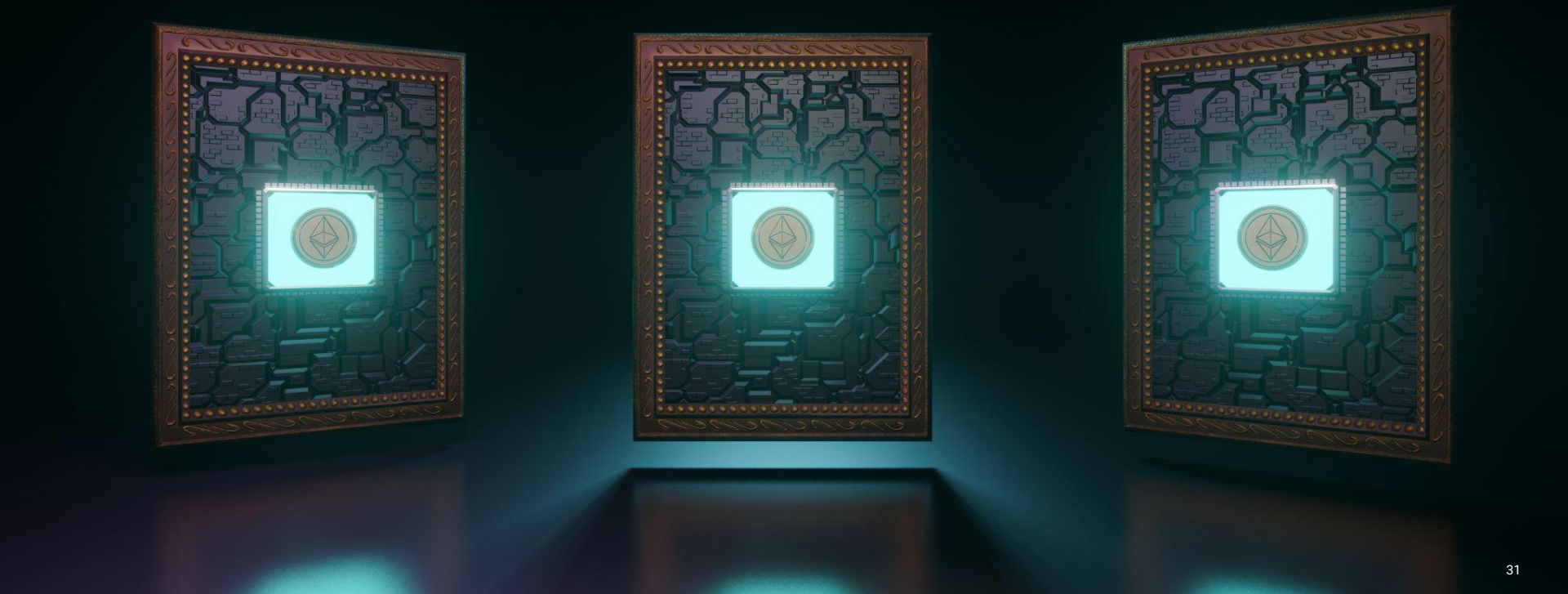
The rules and logic that create and maintain the token get encoded in a token smart contract.



Because we build tokens by using smart contracts, we can program them to do many things besides making payments.

Tokens and Smart Contracts

Tokens make it easier and more efficient to represent the value of an asset and to trade that asset. Representing commodities as tokens can thus simplify the transfer of ownership.



Introduction to Tokenomics

Because tokens are powered by blockchain technology, they can be globally traded without involving traditional financial institutions or physical infrastructure.

Transferring a metric ton of gold requires lots of energy.

Transferring tokens that represent that metric ton requires significantly less energy.

Furthermore, the borderless nature of blockchain means that the transfer won’t have geographical limitations.

Introduction to Tokenomics

Imagine a digital wallet that contains records of all the assets that an individual owns, both digital and physical, in a single place.

They can manage their home ownership, car payments, artwork, investments, video game items, and more with math that cryptographically proves their ownership of these items and that verifies any transactions associated with them.

Introduction to Tokenomics

With this system, businesses can also track and transfer assets in a way that improves their liquidity and auditability.

- For example, food items that get purchased at a grocery store could be tokenized.
- Shoppers could then scan codes on the food items to pull up the entire supply chain history for the purpose of proving exactly where they came from.
- In fact, some companies have already begun using blockchain technology to track food sourcing in this manner.

Coins vs. Tokens

Coins vs. Tokens

Blockchain tokens often represent cryptocurrency. So, some people in the blockchain community use the terms **token** and **coin** interchangeably. However, they represent two similar but distinct concepts.

Coins vs. Tokens

Coins	Tokens
Represent cryptocurrencies that are native to their blockchains, such as Bitcoin and ether.	Represent any tangible or intangible asset of value.
Run on their own blockchain platform.	Created by deploying smart contracts over existing blockchain platforms.
Can be used as a unit of account, or to transfer money, store value, make purchases, etc.	Mostly used with decentralized apps (dApps) or to represent something of value.

Stablecoins are tokens designed to have stable value. Unlike most cryptocurrency coins, they are typically backed by a fiat, or government, currency.

Stablecoins

A stablecoin company often holds a stable currency, like US dollars, in its bank account and then issues tokens that are backed by those dollars.

CURRENCY	MARKET CAPITALIZATION	COLLATERAL TYPE	↗
Tether	$70,135,616,277	Fiat	↗
USD Coin	$30,545,605,242	Fiat	↗
Binance USD	$13,220,410,692	Fiat	↗
Dai	$6,103,401,727	Crypto	↗
TrueUSD	$1,435,012,977	Fiat	↗
PAX Gold	$315,972,930	Precious metals	↗
HUSD	$311,667,716	Fiat	↗
sUSD	$299,484,505	Crypto	↗

Questions?

Building Tokens with Solidity

Think of a mapping as an association between two variables.

Mapping

We can map account balances to account addresses. The developer can thus associate a balance with a specific address and then retrieve the current balance for that account at any time.

```
`mapping(_KeyType => _ValueType)`
```

word = key	definition = value

```
{"python" : "constricting_snake"}
```

Mapping

To understand the usefulness of mapping for building tokens, let's return to the arcade example from earlier in the lesson.

Say that we own the arcade and that we want to switch from a physical-token system to a blockchain-token system.

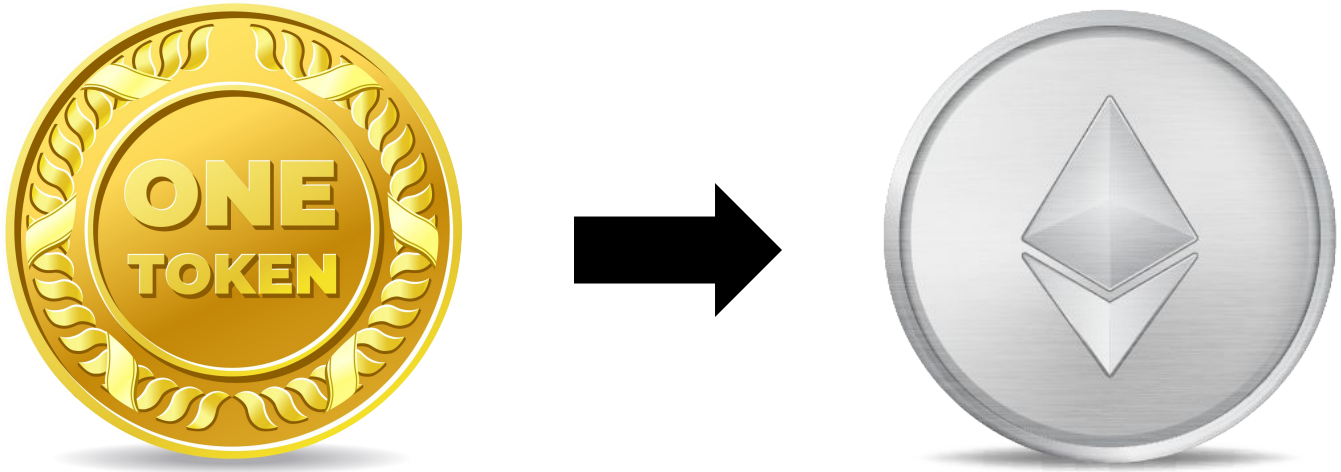
Rather than having customers exchange cash for plastic tokens, we'll have them exchange ether for blockchain tokens.

They can then spend these tokens to play games and redeem prizes at the arcade.

We can map the customer account addresses to their account balances. This way, we can track the number of arcade tokens that any customer has at any time.

Mapping

We want to convert the concept of the arcade with its arcade tokens into a token smart contract.



The token smart contract will contain the logic that creates the tokens, that handles the transactions when arcade customers exchange ether for tokens, and that tracks the token balances in the customer accounts.

We have just one more feature of Solidity that we need to learn about before coding our token: the message object.

The Solidity `msg` Object

- It's a **global object**—it's accessible from any smart contract that we write with Solidity, without defining the object within our code.
- It refers to the current message that's being sent to the blockchain.
- It has several attributes, including `msg.sender` and `msg.value`.

With the concepts of mapping and the message object in mind, it’s time to use Solidity to create arcade tokens!

Instructor Demonstration

Building Tokens with Solidity

Activity: Arcade Token

In this activity, you'll use the mapping data structure to build a token smart contract for use by an arcade—or for any other business that you would like!

Suggested Time:

30 minutes

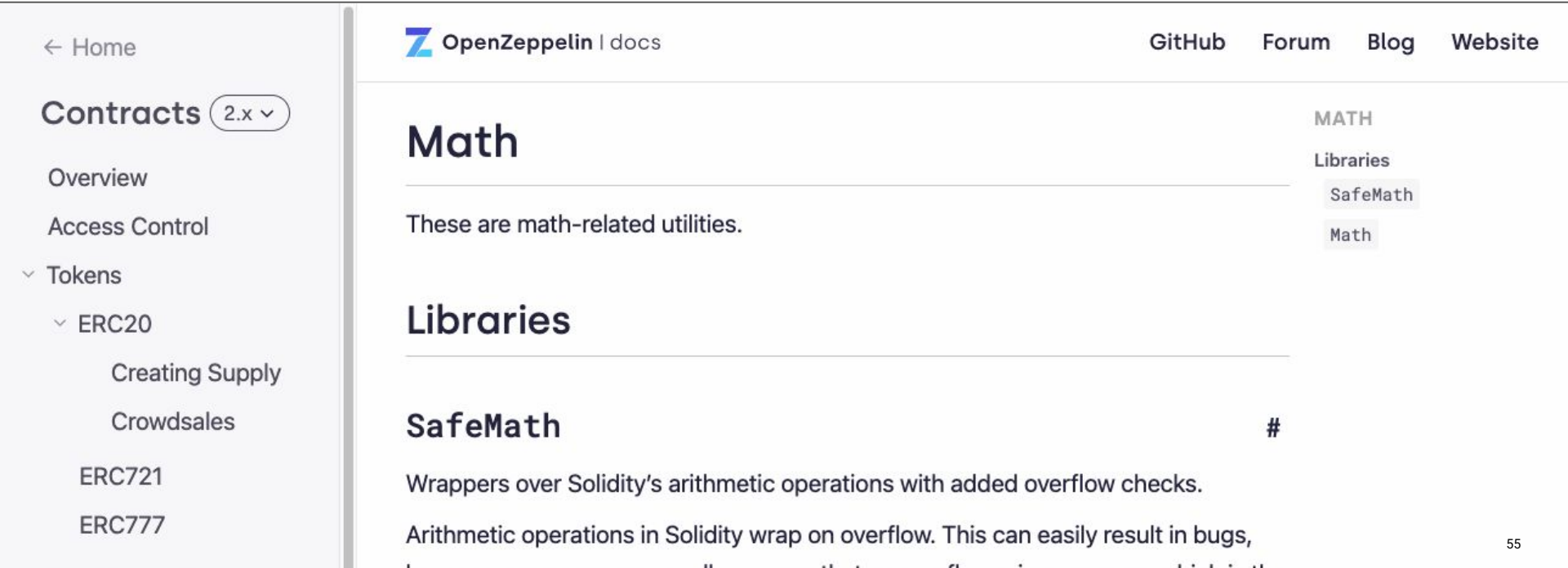
Time's Up! Let's Review.

Questions?

Securing Contracts with Solidity Libraries

SafeMath from Open Zeppelin

Although smart contracts and the resulting tokens are relatively easy to create, we can use third-party libraries—specifically, the [OpenZeppelin SafeMath library]—to make our smart contracts both more secure and more efficient to write.



SafeMath from OpenZeppelin

OpenZeppelin specializes in developing secure smart contracts that use Ethereum community standards. They provide several libraries like SafeMath for smart contract development.

OpenZeppelin includes many standardized smart contracts that blockchain developers can adapt, customize, and build from in order to write more-secure and more-efficient Solidity code.

The arcade token that we created earlier in this lesson aren't secure.
The contract is vulnerable to rewarding an infinite number of tokens to a customer via an integer underflow error. So, we'll use the SafeMath library to make the token more secure.

SafeMath from OpenZeppelin

SafeMath adds special functions to any object using the `uint` type.

Without SafeMath	With SafeMath
+	.add
−	.sub
×	.mul
÷	.div

Integer Overflow Example

Imagine that the odometer on a car has reached the maximum value that it supports. What happens after it reaches Mile 999,999? The odometer runs out of higher numbers and resets back to zero.

An integer underflow is the opposite.

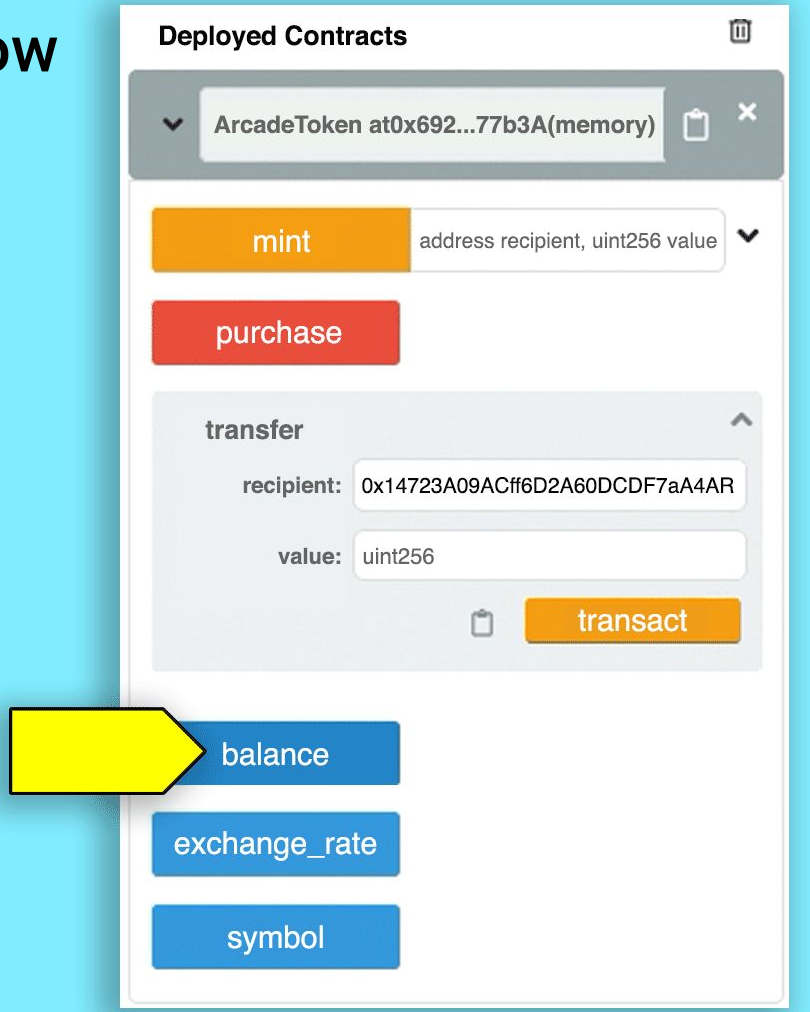
Integer Underflow Example

In the case of ArcadeToken, say that an arcade customer has zero tokens and tries to spend tokens—that is, to subtract tokens from the zero balance.

Instead of getting a negative `uint` balance, the customer gets a new balance that's the highest number a `unit` can hold. That represents an enormous number of tokens!

0 - 0 =

Integer Underflow



Questions?


Break

Time to Code

SafeMath

Suggested Time:

15 minutes

Questions?

Time to Code

Creating an XP-Token

Suggested Time:

30 minutes

Time's Up! Let's Review.

Questions?

Instructor Demonstration

Review Creating an XP-Token

Time's Up! Let's Review.

Questions?

Instructor Demonstration

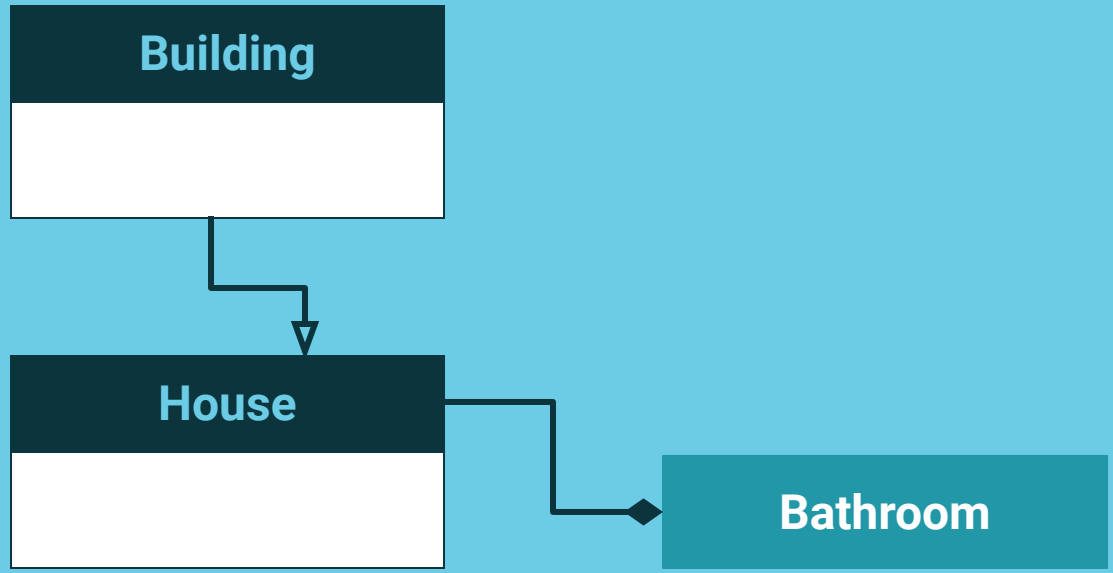
Inheritance and Composition

Inheritance and Composition

Writing smart contracts (like many programming activities) heavily relies on the OOP principles of **inheritance** and **composition**.

“is a”
(inheritance)

“has a”
(composition)

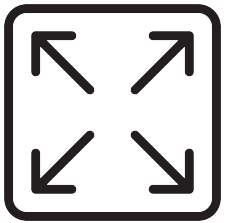


Inheritance and Composition

We will focus on two core targets that we want to achieve by using these principles:

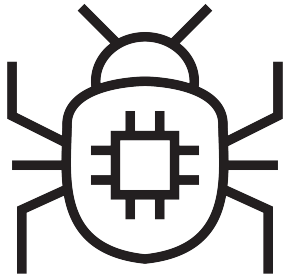
Ability to easily scale or reuse code

Once we or others have designed robust and efficient code, we want to use it for other use cases without having to modify the code.



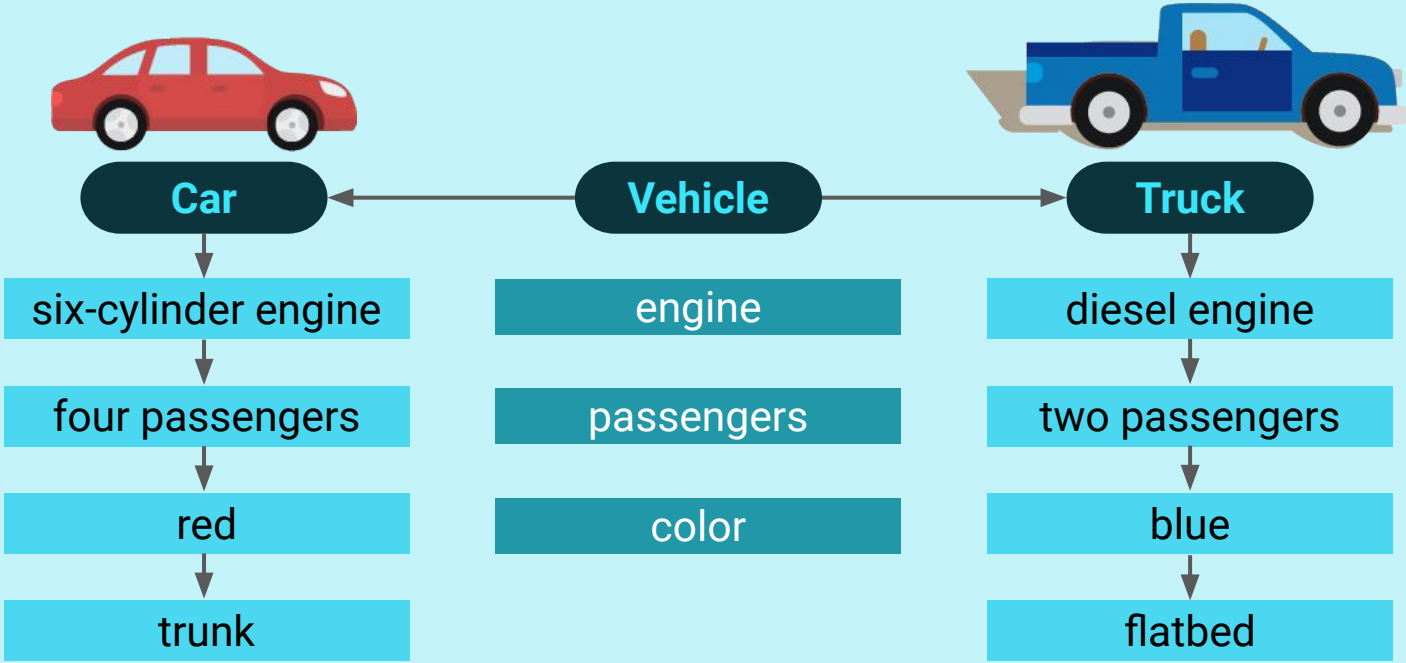
Reduction of the potential for bugs

Being able to reuse well-tested code allows us to become more efficient and reduce the chance for bugs in our code.



Inheritance

In OOP, we use inheritance when a particular type of relationship exists between classes. Specifically, this happens when one class (the **subclass**) is a specialized version of the other, more-general class (the **superclass**).

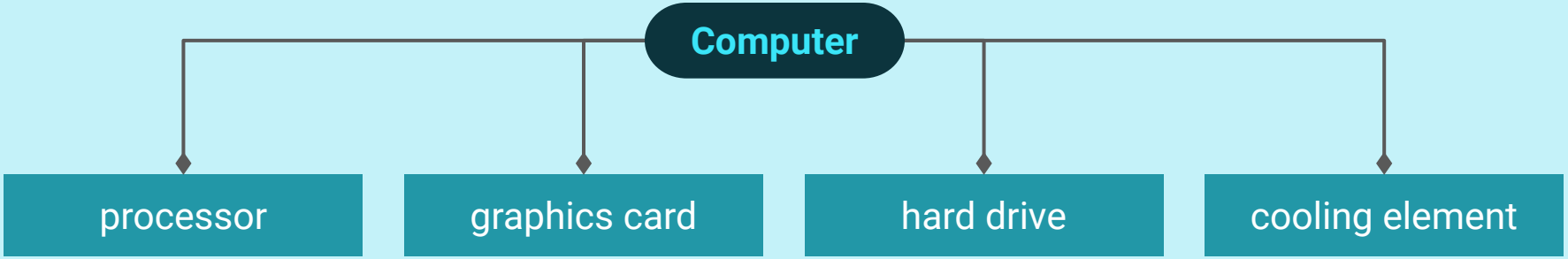


With inheritance, is a is the key phrase.

A truck is a vehicle.

Composition

In OOP, we use composition for cases where a class or object has elements of another class or object.



With composition,
has a is the key phrase.

A computer has a processor.

Which one is the better approach?

As always, the answer is that it depends on the use case.

Inheritance and Composition

Inheritance

- Emphasizes being of a certain type.
- Inheritance makes sense when a large overlap (like 80%) exists between two classes, and one class is a specialized case of the more-general class.

Composition

- Emphasizes the composition of a class or object based on elements of other types.
- Contains elements of other classes that provide the desired functionality.
- Composition is more flexible.
  With composition, we can choose the components that we want to use to construct a new class.

Inheritance and Composition

For the upcoming lessons, we'll use **inheritance** to design our tokens.
This is generally true for most implementations of tokens within smart contracts.
Our tokens will inherit characteristics from tokens that Ethereum standards define and that OpenZeppelin makes available.

OpenZeppelin

ethereum

Questions?


RECAP

What's the solution that we used to fix our underflow and overflow issues?

Answer

We used the OpenZeppelin SafeMath library to fix our underflow and overflow issues.

Who is ultimately responsible for the cybersecurity of an organization?

Answer

Everyone is responsible for carrying the burden of security.

Why not just offload this stuff to the security team?

Answers

The security team is already going to be overwhelmed with many other things to patch and is fighting a constant upward-hill battle.

Every little bit of effort towards security helps.

We can’t be lazy when developing and leave security as an afterthought.

That’s how the technology industry became so vulnerable in the first place!

Questions?


The End