

Mastering Blockchain Programming with Solidity

Write production-ready smart contracts for Ethereum blockchain with Solidity



Packt

www.packt.com

Jitendra Chittoda

Mastering Blockchain Programming with Solidity

Write production-ready smart contracts for Ethereum
blockchain with Solidity

Jitendra Chittoda

Packt>

BIRMINGHAM - MUMBAI

Mastering Blockchain Programming with Solidity

Copyright © 2019 Packt Publishing

All rights reserved. No part of this book may be reproduced, stored in a retrieval system, or transmitted in any form or by any means, without the prior written permission of the publisher, except in the case of brief quotations embedded in critical articles or reviews.

Every effort has been made in the preparation of this book to ensure the accuracy of the information presented. However, the information contained in this book is sold without warranty, either express or implied. Neither the author, nor Packt Publishing or its dealers and distributors, will be held liable for any damages caused or alleged to have been caused directly or indirectly by this book.

Packt Publishing has endeavored to provide trademark information about all of the companies and products mentioned in this book by the appropriate use of capitals. However, Packt Publishing cannot guarantee the accuracy of this information.

Commissioning Editor: Richa Tripathi
Acquisition Editor: Shriram Shekhar
Content Development Editor: Manjusha Mantri
Senior Editor: Afshaan Khan
Technical Editor: Pradeep Sahu
Copy Editor: Safis Editing
Project Coordinator: Prajakta Naik
Proofreader: Safis Editing
Indexer: Rekha Nair
Production Designer: Shraddha Falebhai

First published: August 2019

Production reference: 1010819

Published by Packt Publishing Ltd.
Livery Place
35 Livery Street
Birmingham
B3 2PB, UK.

ISBN 978-1-83921-826-2

www.packtpub.com

I dedicate this book to my wife Neha, and my two daughters Arshiya and Advika with much love.



Packt.com

Subscribe to our online digital library for full access to over 7,000 books and videos, as well as industry leading tools to help you plan your personal development and advance your career. For more information, please visit our website.

Why subscribe?

- Spend less time learning and more time coding with practical eBooks and Videos from over 4,000 industry professionals
- Improve your learning with Skill Plans built especially for you
- Get a free eBook or video every month
- Fully searchable for easy access to vital information
- Copy and paste, print, and bookmark content

Did you know that Packt offers eBook versions of every book published, with PDF and ePub files available? You can upgrade to the eBook version at www.packt.com and as a print book customer, you are entitled to a discount on the eBook copy. Get in touch with us at customercare@packtpub.com for more details.

At www.packt.com, you can also read a collection of free technical articles, sign up for a range of free newsletters, and receive exclusive discounts and offers on Packt books and eBooks.

Foreword

I have known Jitendra Chittoda for a good while. Together, we have audited a significant number of smart contracts for different blockchain platforms. Some of these contracts have been widely used, while others handle large amounts of funds.

Prior to working as a smart contract auditor, Jitendra was working as a developer, which gave him a broader perspective of the field, as he has observed it from a variety of standpoints. These multiple viewpoints of blockchain, and Ethereum in particular, have allowed him more general insights.

While many have heard about the power and potential of blockchain and smart contracts, I believe the blockchain journey is still in its infancy. As with many new technologies, a number of mistakes have been made. Some mistakes led to exploits, and some exploits were so large that they made news headlines. One of those attacks was the DAO reentrancy attack.

Given the novelty of the technology, people require guidance. Such guidance is required on different levels. Firstly, many people still do not sufficiently understand what blockchain is really about, what makes it special, and what differentiates it from regular databases. They don't understand the power of smart contracts—which are sometimes called programmable money—and their potential impact on future businesses.

Secondly, while other people understand the power of blockchain, they do not see a clear path to building on top of it. They are not aware of the many small mistakes that can be made along the way, which range from overly expensive smart contracts to a complete loss of control. Hence, these people require detailed and explicit technical guidance.

Given Jitendra's background, he can provide a good overview and many technical recommendations that will not only speed up learning, but can also avoid costly and potentially project-threatening errors. This book starts off with introductions to blockchain before diving into Solidity, its details, its common applications, its most useful helpers, and its security practices.

Hubert Ritzdorf

Chief Technical Officer, ChainSecurity AG

Contributors

About the author

Jitendra Chittoda is a blockchain security engineer at ChainSecurity. His day job is to perform security audit on smart contracts and expose security vulnerabilities in Solidity and Scilla contracts. He has also developed a non-custodial, decentralized, P2P lending contracts for ETHLend. The Solidity contracts that he has developed or audited handle over \$100 million worth of cryptoassets. He also served as a tech and security advisor in various ICO projects.

Before finding his passion for blockchain, he coded in Java for over 11 years. He is the founder and leader of Delhi-NCR-JUG, a non-profit meetup group for Java. He holds a master's degree in computer applications and is regularly invited as a speaker at various conferences and meetups.

I would like to thank my wife, Neha, for encouraging and supporting me every single day with writing this book. A big shout-out to Manjusha Mantri, Shriram Shekhar, Pradeep Sahu, all of the technical reviewers, and the Packt team for their peer review, suggestions, and full support. I would also like to thank all my mentors and professors who always helped me in becoming a programmer. Thank you, Mom and Dad. You have given me the greatest gift: an education.

About the reviewers

Natalie Chin is a blockchain developer, college professor, and technical blogger. She is currently teaching at George Brown College in Toronto, helping to lead the first blockchain college program in the world. At STK, she built a level-two scaling solution on Ethereum, allowing instant cryptocurrency purchases. Natalie won three prizes at ETHSanFrancisco, 2018, where her team, Lending Party, built a decentralized application allowing a CDP (cryptocurrency loan) to liquidate into a bank account instantly.

She is an avid hackathon organizer, associated with DeltaHacks and Stackathon, where she inspired new developers to join the blockchain space. Natalie is a Women in Tech ambassador and is always passionate about spreading knowledge about the blockchain field.

A sincere thank you to Dad, for introducing me to technology at a young age, supporting me, and perpetually pushing me outside of my comfort zone to grow and be better.

To Zichen Jiang, for always finding the silver lining of situations, and always inspiring me to embrace everything with wit, strength, and humor.

Finally, I am eternally grateful to my mother-in-law, for embodying what it means to live a life of curiosity and positivity.

Ben Weinberg is a self-taught computer programmer who started off in iOS development with Objective-C and Swift, and, eventually, got caught up in the world of blockchain while working in Jerusalem. He eventually started learning blockchain and web development simultaneously with the goal of entering the blockchain space as a developer. Ben worked at the Toronto-based Bitcoin Bay, writing Solidity smart contracts and leading workshops on Ethereum development and Bitcoin opcodes. He also joined George Brown College, where he assisted aspiring blockchain developers as a lab monitor for George Brown's new blockchain development course. At George Brown, he helped teach full-stack web development, smart contract development, and decentralized application development.

I would like to thank all the mentors who have helped me along the way in my endeavors to become a programmer. I would like to acknowledge the Toronto and Jerusalem blockchain communities for being so accessible and welcoming to me as I entered this world without knowing anyone and with little knowledge of how blockchain works. I would also like to send my most supreme thanks to my family for all they have done.

Marc Lijour helps organizations strengthen their competitive advantage with technology such as blockchain and practices such as DevOps. Marc brings consulting experience from the public sector, Cisco, Savoir-faire Linux, and ConsenSys to executives driving innovation. He started the Metamesh Group with 30 consultants spanning across 5 continents. He helped George Brown launch the first blockchain development college program in Canada.

Marc holds degrees in mathematics and computer science, as well as an MBA in technology and innovation. He serves on the board of several not-for-profit organizations, including the **Information and Communications Technology Council (ICTC)**, ColliderX, TechConnex, and the Toronto French Business Network.

Packt is searching for authors like you

If you're interested in becoming an author for Packt, please visit authors.packtpub.com and apply today. We have worked with thousands of developers and tech professionals, just like you, to help them share their insight with the global tech community. You can make a general application, apply for a specific hot topic that we are recruiting an author for, or submit your own idea.

Table of Contents

Preface	1
<hr/>	
Section 1: Getting Started with Blockchain, Ethereum, and Solidity	
<hr/>	
Chapter 1: Introduction to Blockchain	8
Understanding blockchain	9
Blockchain solves the double-spending problem	10
Properties of blockchain	10
Distributed ledger	11
Fault tolerance	11
Attack resistance	11
Remove intermediaries	11
Consensus protocol	11
Faster settlement	12
Lower transaction fees	12
Transparency	12
Immutability	13
Irreversible transactions	13
Trust in the network	13
Artificial trust	13
Trustless systems	14
Availability	14
Empower individuals	14
Chronological order of transactions	14
Timestamped	14
Sealed with cryptography	15
When to use blockchain	15
When not to use blockchain	15
Blockchain is slow	16
Blockchain depends on off-chain oracles	16
Existing implementations of blockchain	17
Cross-border payments	17
Decentralized cloud storage	18
Decentralized computing	18
Introduction to Ethereum	18
Ethereum is for writing decentralized applications	18
Ethereum architecture	18
P2P networks	19
Nodes	19
Full nodes	20
Lightweight nodes	20
Miners	20

Blocks	20
Ethereum Virtual Machine (EVM)	22
Ether currency	22
Smallest unit – wei	23
Gas	23
Gas limit	24
Gas price	24
Formulas	24
Example	25
Ethereum accounts	27
Externally owned accounts	27
Contract accounts	28
The difference between an EOA and a contract	29
Ethereum transaction	30
Transaction fields	30
From	31
To	31
Value	31
Gas limit	31
Gas price	31
Nonce	31
Data	31
Transaction hash	32
Transaction status	32
Pending status	33
Success status	33
Fail status	34
Dropped status	34
Transaction operations	35
Replace/update	35
Cancel	36
Testnets	36
Smart contracts	36
Immutable code	37
Irreversible transactions	37
Think twice before deploying	38
Limited storage	39
Every transaction consumes gas in ether	39
Playing with ether or tokens	39
Summary	40
Questions	41
Chapter 2: Getting Started with Solidity	42
Introduction to the Solidity language	42
The layout of a Solidity source file	44
Solidity version with pragma	44
Importing other source files	44
Structure of a contract	45
Declaring state variables	46
Writing function definitions	46

Creating a custom modifier using function modifiers	47
Using events for logging and callback	48
Custom data types with struct	48
Custom types for constants with enum	49
Solidity data types	50
Understanding Solidity value types	51
Integer value type	51
Boolean value type	52
Address value type	53
Reading a contract's ether balance	54
Sending ether using transfer	55
Sending ether using send	55
Understanding call and delegatecall functions	55
Understanding the staticcall function	57
Adjust gas for a transaction using gas	58
Forwarding ether to another contract	58
Changes in Solidity version 0.5.0	59
Fixed size byte arrays	60
Dynamically sized byte arrays	61
Understanding the bytes data type	61
Understanding the string type	62
Passing functions using function types	62
Get a function type with function selector	63
Using internal function types	63
Using external function types	64
Solidity reference types	66
Understanding variables' data locations in Solidity	66
Using arrays in Solidity	67
Creating a key value map using mapping	68
Resetting variables using the delete keyword	69
Assigning variables with units	69
Specifying ether amounts using ether units	70
Supported units for time	70
Global special variables and functions	71
Using block and transaction properties	71
Getting block information using the block variable	71
Getting sender transaction info using the msg variable	72
Getting the current time using the now variable	72
Getting transaction info using the tx variable	72
Special functions	73
Application Binary Interface encoding functions	73
Error handling in Solidity	74
Cryptographic functions	76
Contract-related functions	76
Get the contract address using this keyword	76
Destroying contracts using the selfdestruct function	77
Destroying contracts using the suicide function	77
Topics for self-study	78
Summary	78

Questions	79
Further reading	79
Chapter 3: Control Structures and Contracts	80
Understanding Solidity control structures	80
Returning multiple values from function	82
Expression evaluation order	82
Solidity contracts	84
Creating contracts	85
Creating child contracts using the new keyword	85
Using variable and function visibility	87
Getter functions for state variables	90
Creating custom function modifiers	91
Creating constant state variables	92
Understanding types of Solidity functions	92
Using view functions to read state variables	93
Using pure functions to perform calculations	95
Using the default fallback function	96
Overloading functions	97
Overriding function definition	98
Using emit and events for event logging	99
Inheriting contracts	100
Passing arguments for the base constructor	103
Understanding inheritance linearization	104
Creating abstract contracts	104
Creating interfaces	106
Creating custom reusable libraries	107
Using libraries with – using...for directive	108
Summary	110
Questions	110
Section 2: Deep Dive into Development Tools	
<hr/>	
Chapter 4: Learning MetaMask and Remix	112
Technical requirements	112
Using the MetaMask plugin	113
Installing and setting up the MetaMask plugin	114
Connecting to different Ethereum networks	119
Getting test ether from faucets	120
Other features of the MetaMask plugin	120
Using the Remix Solidity IDE	120
The Remix IDE overview	121
Compiler tools present under the Compile tab	122
Understanding the Run tab	122
Selecting the environment to connect with	124

Choosing different wallet accounts	124
Transaction parameters	125
Selecting the contract to use	125
Using deploy and attach	125
Deploying a contract	126
Initiating a transaction to execute the function	130
Initiating a call to a view function and state variables	131
Connecting the contract folder using remixd	132
Setting up a local instance of the Remix IDE	134
Using the blockchain explorer at etherscan.io	134
Ethereum wallet software	135
Using myetherwallet.com	136
Summary	137
Questions	137
Chapter 5: Using Ganache and the Truffle Framework	138
Technical requirements	139
Local blockchain with Ganache	139
Starting a local blockchain	141
Creating workspaces for projects	143
Ganache advance configuration	143
The command-line version of Ganache	144
Understanding Truffle framework	147
Setting up a Truffle project	148
Truffle configuration files	149
Configuring the Solidity compiler	149
Configuring networks	151
Choosing which blockchain client to use	152
Personal blockchain client	153
Running blockchain nodes	153
Using Infura	154
Writing contract migration scripts	154
Trigger migration using the migrate option	154
Sample migration script	155
Using artifacts.require() to get the contract instance	155
Using module.exports	156
Deployer	156
Network considerations	157
Available accounts	157
Writing test cases	158
Writing test cases in JavaScript using Mocha	158
Writing test cases in Solidity	160
Debug transactions	161
Summary	163
Questions	163
Chapter 6: Taking Advantage of Code Quality Tools	164

Technical requirements	165
Using the surya tool	165
Installing surya	165
Using surya describe	167
Generating an inheritance graph	168
Generating a function call graph	170
Parsing contracts	171
Generating function traces	172
Listing inheritance dependencies	173
Generating the markdown report	175
Understanding Solidity linters	177
Using the solhint linter	177
Installing the solhint linter	178
Using solhint	178
Using the ethlint linter	180
Installing ethlint	180
Using solium	181
The solidity-coverage tool	183
Installing solidity-coverage	183
Using solidity-coverage	184
Summary	186
Questions	186
<hr/> Section 3: Mastering ERC Standards and Libraries <hr/>	
Chapter 7: ERC20 Token Standard	188
Technical requirements	188
Overview of the ERC20 token standard	189
Use of ERC20 in crowdfunding	190
The motivation behind the ERC20 standard	191
ERC20 standard API	191
ERC20 implementation	192
Contract state variables	193
The balances variable stores account balance	193
The allowed variable stores approved balances	194
The totalSupply variable stores the total token supply	195
The transfer function	196
Difference between the ether and token transfer functions	197
Token transfer does not notify the contact	198
Tokens can be locked	198
The transfer transaction details	199
The approve function	201
Front-running attack on the approve function	202
Preventing a front-running attack	203
The transferFrom function	203
Two-step process for contracts	205

The allowance function	206
The balanceOf function	207
The totalSupply function	207
Events	208
The Transfer event	208
The Approval event	209
Optional functions	209
The name function	210
The symbol function	210
The decimals function	210
Advanced functions	211
The increaseApproval function	211
The decreaseApproval function	212
Summary	213
Questions	214
Chapter 8: ERC721 Non-Fungible Token Standard	215
Technical requirements	216
Overview of the ERC721 NFT standard	216
The ERC721 NFT standard API interface	217
Understanding the ERC721 implementation	219
ERC721 inherits from IERC721 and ERC165	219
ERC721 inherits from ERC165	220
ERC721 inherits from IERC721	221
Understanding ERC721 state variables	221
Token owner mapping kept in <code>_tokenOwner</code>	222
Approved address mapping kept in <code>_tokenApprovals</code>	222
The number of tokens per owner kept in <code>_ownedTokensCount</code>	223
Operator approvals kept in <code>_operatorApprovals</code>	223
The ERC165 interface code for the ERC721, <code>_INTERFACE_ID_ERC721</code>	224
The ERC165 interface code for the ERC721Receiver, <code>_ERC721_RECEIVED</code>	225
The constructor of ERC721	225
The balanceOf function	226
The ownerOf function	226
The approve function	227
The getApproved function	228
The setApprovalForAll function	228
The isApprovedForAll function	229
The transferFrom function	229
The safeTransferFrom function	230
Another safeTransferFrom function	231
The <code>_exists</code> internal function	231
The <code>_isApprovedOrOwner</code> internal function	232
The <code>_mint</code> internal function	233
The <code>_burn</code> internal function	233
Another <code>_burn</code> internal function	234

The _transferFrom internal function	235
The _checkOnERC721Received internal function	236
The _clearApproval internal function	237
Events	238
The Transfer event	238
The Approval event	239
The ApprovalForAll event	240
The ERC721TokenReceiver interface	240
The ERC721Metadata interface	241
The ERC721 enumerable	242
The ERC721 full implementation	243
Summary	243
Questions	244
Chapter 9: Deep Dive into the OpenZeppelin Library	245
Technical requirements	246
The OpenZeppelin project	246
Installation	249
Usage	249
Contract ownership	250
Contract ownership using Ownable.sol	251
Claimable ownership using Claimable.sol	253
Roles library	254
Managing roles using Roles.sol	255
Manage PauserRole using PauserRole.sol	256
Other roles	258
Life cycle contracts	259
Pause/unpause functions using Pausable.sol	259
The ERC20 token standard library	261
ERC20 interface – IERC20.sol	262
Full ERC20 implementation using ERC20.sol	263
Perform safe ERC20 function calls using SafeERC20.sol	265
Create tokens with metadata using DetailedERC20.sol	267
Create mintable tokens using ERC20Mintable.sol	269
Allow token burning using ERC20Burnable.sol	269
Create pausable tokens using ERC20Pausable.sol	270
Math-related libraries	272
Aggregation functions using Math.sol	272
Arithmetic calculation using SafeMath.sol	273
Crowdsale	275
Create crowdsale using Crowdsale.sol	276
Create whitelisted crowdsale using WhitelistCrowdsale.sol	278
Other crowdsale contracts	279
Utility contracts	280
Check for contracts using Address.sol	280

Prevent reentrancy attacks using ReentrancyGuard.sol	281
Summary	282
Questions	283
Chapter 10: Using Multisig Wallets	284
Technical requirements	284
Understanding multisig wallets	285
Benefits of using multisig wallets	288
Precautions when using multisig wallets	288
Learning ConsenSys multisig implementation	289
Setting up your own multisig wallet	293
Deploying your multisig wallet contract	295
Sending ETH from a multisig contract	300
Controlling contracts with multisig	306
Summary	314
Questions	315
Chapter 11: Upgradable Contracts Using ZeppelinOS	316
Technical requirements	317
Understanding upgradable contracts	317
Introduction to ZeppelinOS	318
Creating a ZeppelinOS project	321
Deploying the StableToken contract	323
Upgrading the contract	326
ZeppelinOS commands	331
Precautions while using ZeppelinOS	333
Precautions for state variables	333
Avoid changing the variable declaration order	334
Avoid changing variable data types	334
Avoid adding new variables before existing variables	334
Avoid removing existing state variables	335
Always add new variables at the end	335
Precautions when changing variable names	335
Avoid initial values in the field declaration	336
Precautions for contract inheritance	336
Avoid changing the inheritance order	337
Avoid adding new state variables in base contracts	337
Summary	338
Questions	338
Chapter 12: Building Your Own Token	339
Technical requirements	339
Features of our token	340
Contract architecture design	341
Designing an ERC20 MST token	342
Designing an MST crowdsale contract	344

Setting up the Truffle project	345
Updating configuration	346
Creating Solidity contracts	347
The MSTToken.sol contract	347
The MSTCrowdsale.sol contract	348
Compiling contracts	350
Writing a migration script	351
Running Ganache	353
Running migration scripts	353
Writing test cases	355
Deploying contracts on testnet	357
Infura APIs	357
Updating the configuration	358
Setting up wallet mnemonics	360
Installing dependencies	360
Deploying contracts on testnet	361
Summary	363
Questions	363

Section 4: Design Patterns and Best Practices

Chapter 13: Solidity Design Patterns	365
Security patterns	366
Withdrawal pattern	366
Applicability	368
Access restriction pattern	368
Applicability	370
Emergency stop pattern	370
Applicability	372
Creational patterns	372
Factory contract pattern	373
Applicability	374
Behavioral patterns	374
State machine pattern	375
Applicability	377
Iterable map pattern	377
Applicability	378
Indexed map pattern	378
Applicability	380
Address list pattern	380
Applicability	381
Subscription pattern	381
Applicability	383
Gas economic patterns	383
String equality comparison pattern	384
Applicability	384

Tight variable packing pattern	385
Applicability	388
Life cycle patterns	388
Mortal pattern	388
Applicability	389
Auto deprecate pattern	389
Applicability	390
Summary	391
Questions	391
Chapter 14: Tips, Tricks, and Security Best Practices	392
Technical requirements	393
Smart contracts best practices	393
Avoiding floating pragma	394
Avoid sharing a secret on-chain	394
The commit-and-reveal scheme	396
Be careful while using loops	397
Avoid using tx.origin for authorization	399
Preventing an attack	401
The timestamp can be manipulated by miners	402
The 15-second blocktime rule	402
Carefully making external function calls	403
Avoid dependency on untrusted external calls	404
Avoid using delegatecall to untrusted contract code	404
Rounding errors with division	405
Using assert(), require(), and revert() properly	406
Gas consumption	406
Known attack patterns	406
Front-running attacks	407
Example of an ERC20 approve function	408
Preventing an attack on the approve function	410
Other front-running attacks	411
Reentrancy attacks	412
Preventing a reentrancy attack	414
Replay attack	415
Signature replay attacks	415
Preventing a signature replay attack	416
Integer overflow and underflow attacks	418
Ether can be sent forcibly to a contract	419
Prevention and precaution	420
Security analysis tools	421
Using the Securify tool	421
Summary	424
Questions	425
Assessments	426

Table of Contents

Other Books You May Enjoy	440
Index	443

Preface

Blockchain technology is at its nascent stage. However, technology is slowly moving forward and new developments using blockchain are emerging. This technology has the power to replace trusted third parties with trusted blockchain networks. Bitcoin was the birth of blockchain technology and has shown the world a new peer-to-peer payment system without needing intermediaries. You could say that Bitcoin was the first generation of blockchain technology. However, Ethereum took this innovative technology to the next level—you could call it blockchain generation two—where you can write decentralized applications using smart contracts. Solidity is the most widely used and popular language for writing smart contracts for decentralized applications.

The book starts with explaining blockchain, Ethereum, and Solidity. It mostly focuses on writing production-ready smart contracts in the Solidity language for Ethereum blockchain. It covers basic Solidity language syntax and control structures, and moves on to writing your own contract. It also deep dives into different libraries that you can use while writing contracts. Later on, it covers tools and techniques to write secure, production-ready smart contracts.

Who this book is for

This book is for professional software developers who have started learning blockchain, Ethereum, and the Solidity language and who want to make a career in writing production-ready smart contracts. This book is also aimed at developers who are interested in building decentralized applications over Ethereum blockchain. This book will help you learn the Solidity language for building smart contracts from scratch and make you proficient in building production-ready smart contracts for decentralized applications.

What this book covers

Chapter 1, *Introduction to Blockchain*, starts with how blockchain technology came into existence through the innovation of Bitcoin. This chapter discusses the different properties of a blockchain. It also introduces Ethereum and how it is different from the Bitcoin blockchain. Later, this chapter introduces smart contracts.

Chapter 2, *Getting Started with Solidity*, starts with the basic Solidity language syntaxes and the structure of a contract. You will learn about the different data types available in Solidity. This chapter also discusses globally available variables present in the Solidity language that you can use while writing your smart contract.

Chapter 3, *Control Structures and Contracts*, deep dives into the control structures of Solidity contracts, the different types of functions supported, contract inheritance, and event logging.

Chapter 4, *Learning MetaMask and Remix*, discusses setting up your MetaMask plugin and creating wallets using it. This chapter also discusses using the online Remix IDE to create, compile, deploy, and interact with your Solidity contracts.

Chapter 5, *Using Ganache and the Truffle Framework*, discusses installing and setting up your local blockchain instance using Ganache. This chapter also discusses installing and setting up the Truffle framework, learning how to use its commands, setting up your new Truffle project, writing migration scripts for the Truffle framework, and adding test cases to a project.

Chapter 6, *Taking Advantage of Code Quality Tools*, discusses improving the quality of your contracts by using open source tools such as `surya`, which helps in generating different kinds of reports. This chapter also discusses using Solidity linters to lint your code and fix common issues present in your contracts, as well as running Solidity coverage tools to generate code coverage reports.

Chapter 7, *ERC20 Token Standard*, covers the introduction of the ERC20 token standard. This chapter deep dives into its full implementation details, provides an in-depth study of each function of the standard, and covers different events, optional functions, and a number of advanced functions.

Chapter 8, *ERC721 Non-Fungible Token Standard*, starts with an introduction to the ERC721 standard and the difference between the ERC20 and ERC721 standards. This chapter deep dives into each of the state variables, functions, and events associated with ERC721 implementation. This chapter also discusses some other optional contracts that can be used with the ERC721 standard.

Chapter 9, *Deep Dive into the OpenZeppelin Library*, starts with an introduction to OpenZeppelin library contracts. This chapter will help you learn how to install and use OpenZeppelin library contracts in your Truffle project. It also provides in-depth studies of library contracts such as `Ownable`, `Claimable`, `Roles`, `PauserRole`, `Pausable`, `ERC20`, `SafeERC20`, `DetailedERC20`, `ERC20Mintable`, `ERC20Burnable`, `ERC20Pausable`, `Math`, `SafeMath`, `Crowdsale`, `Address`, and `ReentrancyGuard`.

Chapter 10, *Using Multisig Wallets*, The multisig wallets are special contracts that require multiple signatures to execute a transaction. This chapter provides an introduction to multisig contracts and their usage. This chapter also covers installing, creating, and setting up your own multisig wallet, as well as how to use and control it.

Chapter 11, *Upgradable Contracts Using ZeppelinOS*, provides an introduction to the ZeppelinOS development platform. Topics covered include creating a new project using `zos` commands, creating an upgradable contract, deploying, re-deploying to upgrade the contract, and some precautions to take when using ZeppelinOS for writing upgradable contracts.

Chapter 12, *Building Your Own Token*, helps you learn how to create your own ERC20 token contract from scratch. You will learn how to draft the specification of a contract, set up a Truffle project, create a contract, choose which OpenZeppelin libraries to use, write migration scripts, write test cases, execute migration scripts and test cases in Ganache, and finally, test your contracts on a testnet such as Rinkeby.

Chapter 13, *Solidity Design Patterns*, introduces different Solidity design patterns. These are divided into five categories: *Security patterns*: Withdrawal, Access restriction, and Emergency stop; *Creational patterns*: Factory pattern; *Behavioral patterns*: State machine, Iterable map, Indexed map, Address list, and Subscription; *Gas economic patterns*: String comparison and Tight variable packing; and *Life cycle patterns*: Mortal and Auto deprecate pattern.

Chapter 14, *Tips, Tricks, and Security Best Practices*, helps you to learn different Solidity smart contract best practices, such as avoiding floating pragma, the 15-second blocktime rule, rounding errors, and gas consumption. It also helps you to learn about different security attacks, such as front-running, reentrancy, signature replay, integer overflow and underflow, and how to prevent these attack patterns.

To get the most out of this book

You should have knowledge of any of the existing programming languages. Solidity language syntaxes are very similar to Java, JavaScript, and Python syntaxes. A developer who has created projects using Java, JavaScript, or Python can pick up and learn Solidity very easily. You should also have some basic knowledge of OOPS concepts, blockchain, wallets, private key, public key, consensus algorithms, and exchanges.

One very important thing to note here is that, in order to write smart contracts for decentralized applications, you need to have a completely new mindset. Up until now, you have probably been building applications using Java, JS, or another language in which you can fix bugs later on, even in the production environment when something goes wrong. However, smart contracts are immutable. If your contract is deployed in production, you are done. If there is a bug left in the contract, you will be unable to fix it. Hence, you need to ensure that all your smart contracts are well-tested and have no bugs present in them.

Download the example code files

You can download the example code files for this book from your account at www.packt.com. If you purchased this book elsewhere, you can visit www.packt.com/support and register to have the files emailed directly to you.

You can download the code files by following these steps:

1. Log in or register at www.packt.com.
2. Select the **SUPPORT** tab.
3. Click on **Code Downloads & Errata**.
4. Enter the name of the book in the **Search** box and follow the onscreen instructions.

Once the file is downloaded, please make sure that you unzip or extract the folder using the latest versions of the following:

- WinRAR/7-Zip for Windows
- Zipeg/iZip/UnRarX for Mac
- 7-Zip/PeaZip for Linux

The code bundle for the book is also hosted on GitHub at <https://github.com/PacktPublishing/Mastering-Blockchain-Programming-with-Solidity>. If there's an update to the code, it will be updated on the existing GitHub repository.

We also have other code bundles from our rich catalog of books and videos available at <https://github.com/PacktPublishing/>. Check them out!

Download the color images

We provide a PDF file that has color images of the screenshots/diagrams used in this book. You can download it at https://static.packt-cdn.com/downloads/9781839218262_ColorImages.pdf.

Code in action

To see the code being executed please visit the following link: <http://bit.ly/2Yv6kpm>.

Conventions used

There are a number of text conventions used throughout this book.

CodeInText: Indicates code words in text, database table names, folder names, filenames, file extensions, pathnames, dummy URLs, user input, and Twitter handles. Here is an example: "Solidity supports different data types, including `uint`, `int`, `address`, and many more."

A block of code is set as follows:

```
contract VariableStorage {
    uint storeUint; //uint256 storage variable
    //...
}
```

When we wish to draw your attention to a particular part of a code block, the relevant lines or items are set in bold:

```
contract VariableStorage {
    uint storeUint; //uint256 storage variable
    //...
}
```

Any command-line input or output is written as follows:

```
$ npm install -g ganache-cli
```

Bold: Indicates a new term, an important word, or words that you see on screen. For example, words in menus or dialog boxes appear in the text like this. Here is an example: "As mentioned earlier, you can start a local Ganache blockchain instance by just clicking on the **QUICKSTART** button."



Warnings or important notes appear like this.