Kapitel 2
Einführung

In diesem Kapitel werden Sie an das Thema Blockchain herangeführt. Wir erklären, was die Blockchain ist, wie sie sich weiterentwickelt hat und wofür die Technologie genutzt werden kann.

Die Technologie Blockchain hat es in wenigen Jahren geschafft, eine immense Aufmerksamkeit auf sich zu ziehen. Vor allem Kryptowährungen, die durch eine Blockchain realisiert werden, rückten mit ihren explosionsartigen Kursanstiegen in das mediale Rampenlicht. Nicht nur Informatiker und Technikfans, sondern auch Normalverbraucher und Firmenvertreter drängen sich derzeit auf Blockchain-Vorträge, um dort ihre Fragen beantwortet zu bekommen: *Was ist eigentlich diese Blockchain, von der plötzlich alle reden?* Und vielleicht noch viel wichtiger: *Wofür kann ich die Blockchain überhaupt gebrauchen?* In diesem Kapitel gehen wir diesen Fragen auf den Grund. Wir erklären zuerst, was sich hinter dem Begriff *Blockchain* verbirgt und warum es sie gibt. Anschließend gehen wir auf die Geschichte der Blockchain ein und für welche Anwendungsfälle sich die neue Technologie eignet.

2.1 Was ist die Blockchain?

Am Anfang jeder Lösung steht idealerweise eines oder mehrere Probleme. Das Hauptziel der Blockchain ist, grundlegende Herausforderungen des Internets zu lösen. Diese möchten wir Ihnen im Folgenden kurz vorstellen. Anschließend zeigen wir Ihnen, wie die Blockchain diese Herausforderungen angeht.

2.1.1 Herausforderungen des Internets

Das Internet erleichtert seinen Anwendern in vielerlei Hinsicht das Leben und löst zahlreiche Herausforderungen der realen Welt. Sie können Entfernungen überbrücken, indem Sie Freunden in aller Welt in Sekundenschnelle Katzenbilder schicken. Sie können rund um die Uhr in Onlineshops einkaufen, online Geld überweisen oder stundenlang Lieblingsserien streamen. Das Internet bringt allerdings auch Herausforderungen und Nachteile mit sich, die uns aus der realen Welt in dieser Form so nicht bekannt sind.

Kurz gesagt, hier geht es um die Herausforderung der *Zentralisierung* und die Herausforderung des *Vertrauens* sowie um das *Double-Spending-Problem*.

Zentralisierung

Als mit dem *Arpanet* 1969 der Vorläufer des Internets in Betrieb genommen wurde, sollte damit vor allem ein dezentrales Netz geschaffen werden, das möglichst ausfallsicher ist. Auch als das Internet später der breiten Masse zugänglich wurde, behielt es seinen dezentralen Charakter. Während der Nutzer anfänglich hauptsächlich Inhalte konsumierte, indem er beispielsweise einen Artikel las, hatte er mit dem Beginn des Web 2.0 verstärkt die Möglichkeit, selbst aktiv am Geschehen im Netzwerk teilzuhaben. Hiermit begann auch eine Art Zentralisierung des dezentralen Netzwerks. Große Firmen wie Facebook, Google oder Amazon zentralisieren Nutzer auf ihrer jeweiligen Plattform und beherbergen auf ihren Servern einen großen Teil des Datenverkehrs und der Daten im Internet. Diese Entwicklung bringt Gefahren mit sich.

Während große Firmen noch vor einiger Zeit eigene Server unterhielten, setzen mittlerweile viele auf Serveranbieter wie *Amazon Web Services* (AWS). Technische Probleme bei AWS können also mit Leichtigkeit die Stabilität des ganzen Netzes ins Wanken bringen. Vorfälle in der Vergangenheit haben bereits gezeigt, dass dieses Szenario nicht abwegig ist.

Die Plattform Facebook wird heutzutage, als größtes soziales Netzwerk, von mehreren Milliarden Nutzern zum Austausch von Informationen und zur Kommunikation genutzt. Diese enorme Nutzerzahl gibt der Firma gleichzeitig auch eine große Macht. Ohne Weiteres können Informationen auf der Plattform zensiert oder Nutzer durch Sperrung ausgeschlossen werden.

Die Zentralisierung macht Informationen im Internet also angreifbarer für Verlust, Manipulation und Zensur.

Vertrauen

Mangelndes Vertrauen zwischen zwei oder mehreren Parteien ist ein Problem, das auch in unserer realen Welt abseits des Internets besteht. Die im Netz vorherrschende Anonymität macht das Fassen von Vertrauen jedoch noch deutlich schwerer. Sei es bei einem Kauf auf der Plattform eBay, dem Verkauf eines Autos oder dem Abschluss eines Vertrags: Aufgrund des fehlenden persönlichen Kontakts ist es für die Vertragspartner schwierig, das Gegenüber einzuschätzen. *Bezahlt mich der Käufer wirklich? Will mich die andere Seite reinlegen? Ist die andere Seite wirklich die Person oder Firma, die sie vorgibt zu sein?* Bei realen zwischenmenschlichen Kontakten können Sie sich hierbei oft auf die eigene Intuition, das Bauchgefühl, verlassen. Kommt Ihnen der Vertragspartner zwielichtig vor, werden Sie sich kaum auf einen Handel mit ihm einlassen. Dieser persönliche Kontakt fällt im Internet jedoch weg.

Eine Lösungsmöglichkeit für dieses Anonymitätsproblem besteht darin, eine weitere Instanz als Mittelsmann hinzuzuziehen, der beide Parteien vertrauen. Klassischerweise übernehmen zentralisierte Dienste, sogenannte *Trusted Third Parties* (TTP) diese Aufgabe. Beispiele für solche TTPs sind Zeitstempeldienste bzw. TSA (*Time Stamp Authorities*) für zeitkritische Dokumente oder Zertifizierungsstellen für digitale Zertifikate. Sie helfen dabei, die Kommunikation sowie Transaktionen zwischen verschiedenen Parteien abzusichern, indem sie als unparteiischer Vermittler vermerken, ob eine E-Mail pünktlich verschickt wurde, oder einen öffentlichen Schlüssel einer Organisation zuzuordnen. Banken oder Kreditkartenanbieter stellen wohl die bekanntesten TTPs dar, die uns im Alltag begegnen. Sie stellen Vertrauen zwischen Parteien her, indem sie garantieren, dass Waren tatsächlich bezahlt werden. Wenn Sie mit Ihrer Kreditkarte eine Bahnkarte online kaufen, kann sich die Deutsche Bahn nach einer kurzen Anfrage bei der TTP darauf verlassen, dass Sie auch wirklich ihr Geld bekommt, denn Ihre Bank bürgt für Sie, indem sie die Überweisung garantiert und sich das Geld später von Ihrem Konto zurückholt.

Uns begegnen im alltäglichen Umgang mit dem Internet noch viele weitere Dienste, die dazu dienen, Vertrauen zu schaffen. Die Verkäuferbewertungen auf Plattformen wie eBay, der Zahlungsanbieter PayPal oder das Bewertungsportal Trustpilot sorgen alle dafür, dass skeptische Kunden einen Zwischendienst einschalten können, der Vertrauen zwischen unbekannten Teilnehmern schafft.

Doch auch sie können diese Vertrauensherausforderung nicht gänzlich lösen. Mit gestohlenen Verkäuferkonten oder gefälschten Bewertungen können Betrüger dem Nutzer beispielsweise recht einfach eine falsche Reputation vorspiegeln. Ein weiteres Problem spricht Satoshi Nakamoto in seinem Whitepaper zu Bitcoin an: Es ist nicht möglich, wirklich unwiderrufliche Zahlungen auszuführen, da die jeweilige TTP im Zweifelsfall als Vermittler eingreifen muss. Ein Kunde könnte die Zahlung relativ einfach zurückfordern. Dies sorgt für Kosten und schafft einen gewissen Grad an Unsicherheit für die Partei, die die Zahlung empfängt. Diese Tatsache führt zu einem erneuten Vertrauensvakuum. Außerdem stellt sich die Frage, wie vertrauensvoll eine TTP selbst ist. Durch die Finanzkrise, beginnend im Jahr 2007, haben viele Menschen das Vertrauen in die Banken verloren. Das Problem ähnelt dem bei der Zentralisierung bereits beschriebenen. Eine zentrale Autorität ist auch immer anfällig für Manipulationen oder Betrug. Es ist kein Zufall, dass die Erfindung der ersten Kryptowährung Bitcoin mit eben dieser Finanzkrise zusammenfällt.

Wer ist eigentlich Satoshi Nakamoto?

Im Jahr 2008 wurde über die Mailingliste für Kryptografie ein Whitepaper mit dem Titel »Bitcoin: A Peer-to-Peer Electronic Cash System« verschickt. Dieses Whitepaper stellte das Konzept der späteren Kryptowährung Bitcoin das erste Mal einer breiteren Personengruppe vor. Verfasst wurde die Arbeit unter dem Pseudonym *Satoshi*

Nakamoto. Bis heute ist allerdings ungeklärt, welche Person oder Personengruppe hinter diesem Namen steckt. Nakamoto nahm die Bitcoin-Blockchain im Januar 2009 in Betrieb und trat unter seinem Pseudonym oder mit seiner E-Mail-Adresse *satoshin@gmx.com* auch in verschiedenen Foren in Erscheinung. Seit einigen Jahren ist Nakamoto allerdings untergetaucht. Viele Gerüchte ranken sich um die Identität hinter dem Pseudonym. Einige Theorien verdächtigen Pioniere der Kryptografie, andere auch Elon Musk als Erfinder von Bitcoin.

In manchen Fällen geht es nicht ohne eine Zwischeninstanz aus der realen Welt. Ein Notar ist ein Garant für die Glaubhaftigkeit von Rechtsgeschäften zwischen Parteien und in manchen Anwendungsfällen, wie etwa dem Grundstückskauf, sogar gesetzlich vorgeschrieben. Durch die hohen Gebühren ist das erkaufte Vertrauen hier vergleichsweise teuer.

Double-Spending-Problem

Das Double-Spending-Problem ist ein Problem, das erst durch die Etablierung von digitalen Strukturen entstanden ist. In unserer realen Welt ist jedes Objekt einzigartig. Wenn Sie einen Apfel essen, ist er weg, und Sie können ihn nicht noch einmal essen. Wenn Sie einen Blumenstrauß verschenken, Ihr Auto verkaufen oder einen 10-Euro-Schein ausgeben, gehen diese Objekte aus Ihrem Besitz und entziehen sich damit Ihrer Handhabe. Dieser Grundsatz gilt in der digitalen Welt nicht. Ein abgespeicherter Film auf der Festplatte kann einfach kopiert oder eine gespeicherte Fotografie beliebig oft verschickt werden. Egal wie oft eine Datei heruntergeladen wird – eine Kopie bleibt immer auf dem Server und damit im Besitz des Herausgebers.

Dies macht die Etablierung digitaler Wertgegenstände oder digitaler Güter schwer. Die inflationären Vervielfältigungsmöglichkeiten verhindern zudem eine Wertentwicklung. Nur eine zentrale Instanz wie beispielsweise ein Lizenzserver kann helfen, indem das Kopieren eingeschränkt wird. Auch Banken behelfen sich mit zentralisierten Rechenzentren, um die Kontostände der Bankkunden zu verwalten und Überweisungen durchzuführen. Ansonsten könnten die als Geburtstagsgeschenk erhaltenen 100 Euro beliebig oft digital überwiesen werden. Natürlich scheint das auf den ersten Gedanken verführerisch. Allerdings würde eine explosionsartige Inflation dafür sorgen, dass unser Geld in Zukunft gar nichts mehr wert wäre.

Um zu verstehen, wie die Blockchain diese Herausforderungen löst, ist es an der Zeit, dass wir uns die Technologie im nächsten Abschnitt genauer ansehen.

2.1.2 Die Blockchain

Die im vorhergehenden Abschnitt geschilderten Herausforderungen und Probleme sind der Hintergrund, vor dem die Entwicklung der Blockchain-Technologie stattfand. Die besonderen Eigenschaften der Blockchain haben das Potenzial, diese Nachteile des Internets zu beheben. Im folgenden Abschnitt erhalten Sie einen Überblick darüber, wie das technisch funktioniert.

Wie bei neuen Technologien üblich, gibt es zahlreiche Definitionen, die das jeweilige Thema aus verschiedenen Blickwinkeln betrachten. Eine allgemeingültige Definition hat sich für die Blockchain-Technologie noch nicht durchgesetzt.

Meistens wird die Blockchain als *Datenbank*, *Protokoll*, *Logfile*, *Datenstruktur* oder *Ledger* (zu Deutsch *Hauptbuch*) bezeichnet, oft in Verbindung mit den Adjektiven *verteilt* oder *dezentral*. Die Autoren Condos, Sorrell und Donegan beschreiben die Blockchain in ihrer Veröffentlichung »Blockchain Technology: Opportunities and Risks« als elektronisches Hauptbuch (*Register*), bestehend aus digitalen Transaktionen, Datensätzen oder Ereignissen, die als *Hash* zusammengefasst sind, um Sicherheit zu gewährleisten. Dieses Hauptbuch wird durch ein verteiltes Netzwerk aus Teilnehmern beglaubigt und verwaltet, indem ein sogenanntes *Konsensprotokoll* zum Einsatz kommt.

Wenn wir die Blockchain-Technologie laut dieser Definition betrachten, ist es also hilfreich, zwischen zwei Teilen der Technologie zu differenzieren: einer Datenstruktur (nämlich der Blockchain) sowie dem System, das diese Datenstruktur verwaltet (dem Blockchain-System). Ein Blockchain-System besteht aus vielen verteilten Knoten, die in einem Netzwerk zusammengeschlossen sind. Solche Knoten könnten beispielsweise Ihr heimischer PC, ein Server in Island oder ein Notebook in Asien sein.

Je mehr Knoten es gibt und je größer und weiter ihre Verteilung ist, desto besser ist das für die jeweilige Blockchain. Denn je größer das Netz der einzelnen Knoten ist, desto besser wird die Herausforderung der Zentralisierung angegangen. Als Teilnehmer des Netzwerks ist nämlich jeder Knoten stolzer Inhaber einer Kopie der gesamten Blockchain. Es werden also wirklich alle Daten der Blockchain auf jedem einzelnen Knoten verteilt gespeichert.

Während bei der Datenhaltung in zentralisierten Systemen versucht wird, Redundanzen möglichst aufzulösen, ist die redundante Speicherung der gesamten Datenbank ein wesentliches Sicherheitsmerkmal der Blockchain-Technologie. Die Gefahren von klassischen Netzwerken haben wir bereits anhand der Herausforderung der Zentralisierung vorgestellt. Durch die Verteilung in der Blockchain werden Verlust, Manipulation und Zensur, die durch eine zu hohe Macht einzelner Instanzen entstehen, verhindert. Einige Knoten in einem Gebiet werden durch ein Hochwasser zerstört? Das bringt die Blockchain nicht ins Wanken, da noch Tausende weiterer Knoten eine Kopie der Blockchain besitzen. Ein korruptes Staatsoberhaupt will bestimmte Daten löschen lassen oder abändern? Es wird ihm keinesfalls gelingen, alle Kopien auf der Welt zu manipulieren. Sicher fragen Sie sich jetzt, wie Sie wissen sollen, welche Kopie der Blockchain dann die richtige und valide Version ist? Dies

festzustellen, ist eine Aufgabe des gesamten Netzwerks und des Systems selbst. Die Funktionen, die dafür sorgen, werden wir Ihnen in Kapitel 3 noch genauer vorstellen.

Eine weitere Aufgabe der Blockchain ist es, diese große Anzahl an Kopien aktuell zu halten und auch Schreiboperationen von neuen Datensätzen in die Blockchain auf ihre Gültigkeit zu prüfen. Diese Aufgabe wird als die Findung eines *Konsens*, also einer Übereinstimmung im Netzwerk, bezeichnet. Zur Konsensfindung müssen die menschlichen Eigentümer der Knoten dazu motiviert werden, ihre Ressourcen zur Verfügung zu stellen. Warum sollten Sie Ihren PC den ganzen Tag laufen lassen, nur um es jemandem auf der anderen Seite der Welt zu ermöglichen, eine Überweisung ohne Bank als Mittelsmann durchzuführen? Einen Anreiz bietet die Blockchain normalerweise mit der Ausschüttung von Kryptowährungen als Belohnung an die Helfer. Der Prozess der Konsensfindung (oder zumindest ein Teil davon) wird auch *Mining* genannt und ebenfalls in Kapitel 3 genauer vorgestellt. Weiterhin muss in der Blockchain sichergestellt werden, dass Informationen, wie etwa die Transaktion von Einheiten einer Kryptowährung, nur von dazu berechtigten Personen in Umlauf gebracht werden dürfen. Im Fall der Kryptowährung wären dies die Besitzer der jeweiligen Einheiten. Außerdem muss die Blockchain selbst so aufgebaut sein, dass sie möglichst einfach zu überprüfen und möglichst schwer zu manipulieren ist. Dieser Anspruch richtet sich aber vor allem an die Datenstruktur der Blockchain.

Die Blockchain kapselt die ihr anvertrauten Informationen (*Transaktionen*, *Datensätze* oder *Ereignisse*) in Blöcken, die Sie beispielhaft innerhalb der einzelnen Knoten in Abbildung 2.1 sehen.

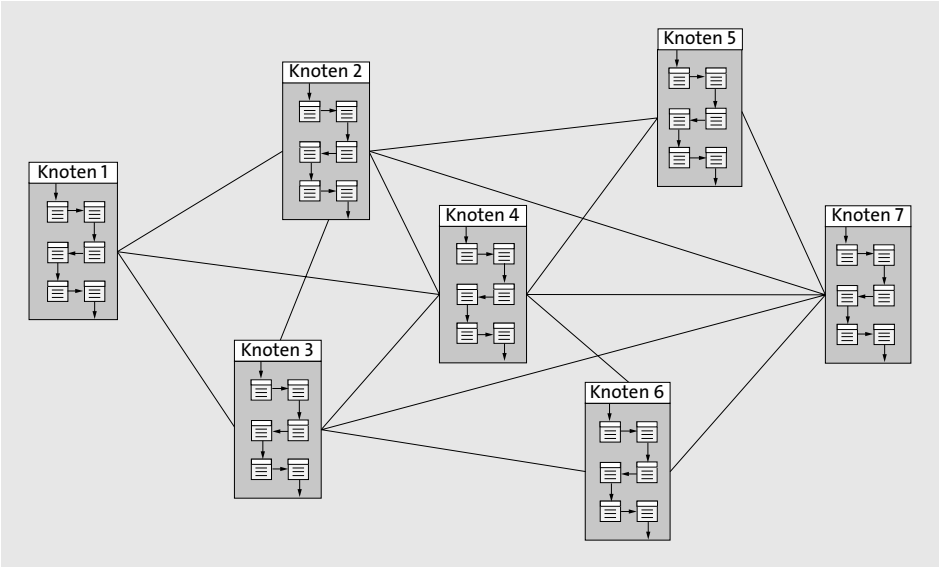

Abbildung 2.1 Vereinfachte Übersicht eines Blockchain-Netzwerks

Diese kompakten Pakete können vom Netzwerk einfach geprüft werden. Ein geprüfter Block wird versiegelt und der Blockchain hinzugefügt. Die Blöcke sind untereinander verkettet, wie es sich für eine richtige Kette gehört. Bei der Verkettung wird sichergestellt, dass alle Blöcke aufeinander aufbauen und kleinste Änderungen im gesamten Konstrukt bei der Überprüfung sofort auffallen würden. Dafür verwendet die Technologie verschiedene kryptografische Verfahren – auch die werden wir Ihnen in Kapitel 3 genauer vorstellen.

Blockchain oder Distributed Ledger

Der Begriff *Blockchain* wird in der Literatur häufig synonym mit dem Begriff *Distributed Ledger* (*verteiltes Hauptbuch*) verwendet. Beide Konzepte sind sich zwar sehr ähnlich, unterscheiden sich aber in einem bestimmten Punkt: Während die Blockchain die Datensätze in Blöcken sortiert, werden die Datensätze beim Distributed Ledger einfach aufeinanderfolgend abgelegt (Walport, 2015). Verstehen Sie die Blockchain also als eine besondere Ausprägung eines Distributed Ledger.

Damit die einzelnen Teilnehmer durch Transaktionen, Datensätze oder Ereignisse mit dem Netzwerk interagieren können, benötigen sie Adressen. Zudem müssen sie beweisen, dass sie wirklich der rechtmäßige Inhaber dieser Adresse sind. Dies wird in der Blockchain durch asymmetrische Kryptografie bewerkstelligt. Das System generiert dem Teilnehmer einen zufälligen Private Key und berechnet daraus den zugehörigen Public Key. Besonders der Private Key ist eine wichtige Instanz in der Blockchain. Dieser Schlüssel ist (im Idealfall) nur dem jeweiligen Teilnehmer bekannt und verhält sich wie eine Art PIN oder ein Passwort. Um eine Aktion im Netzwerk zu tätigen, muss der Teilnehmer diese mit seinem Private Key signieren. Besitzt ein anderer Teilnehmer den öffentlichen Public Key, kann er diese Signatur verifizieren. Aus dem Public Key des Nutzers wird durch ein Hashverfahren auch die Adresse des Teilnehmers berechnet, die so etwas wie eine Kontonummer im Netzwerk darstellt. Um eine Nachverfolgung von Aktionen im Bitcoin-Netzwerk auszuschließen, wird Nutzern häufig dazu geraten, für jede Transaktion eine neue Adresse zu generieren. Hierbei geht schnell der Überblick verloren. In einem sogenannten *Wallet* können die Key-Paare für verschiedene Adressen aufbewahrt werden.

Wallets

In Wallets werden klassischerweise die Private Keys in Kombination mit den Public Keys eines Nutzers abgelegt. Es gibt Wallets in verschiedenen Ausprägungen: *Paper Wallets*, *Hardware Wallets*, *Software Wallets* und *Website Wallets* (Liu et al., 2017). Bei Paper Wallets werden sowohl der Private Key als auch der Public Key auf Papier ausgedruckt. Der Private Key ist dabei oft als QR-Code abgebildet. In Hardware Wallets wird der Private Key langfristig auf einem Hardwaregerät gespeichert. Dieses Gerät

kann mit einem Computer verbunden werden. Software und Website Wallets erlauben dem Nutzer, mit einer aufbereiteten Nutzeroberfläche mit der Blockchain zu interagieren. Bei Software Wallets ist es notwendig, ein Programm auf dem eigenen Rechner zu installieren. Die Website Wallets können über den Browser aufgerufen werden. Um ein Website Wallet zu verwenden, muss der Nutzer seinen Private Key eingeben, kann aber auch ein Hardware Wallet nutzen. Sowohl in Software als auch in Website Wallets können die Nutzer, je nach Anbieter, ihre Transaktionshistorie oder ihr Guthaben in der jeweiligen Kryptowährung einsehen, Transaktionen verschicken oder andere Services in Anspruch nehmen.

2.1.3 Die Blockchain als Problemlöser

Mit ihrem besonderen Aufbau widmet sich die Blockchain geschickt den Herausforderungen des Internets aus Abschnitt 2.1.1. Dank der redundanten Verteilung wird einer Zentralisierung entgegengewirkt, denn alle Knoten im Netzwerk besitzen die Daten und sorgen für deren Richtigkeit. Keine zentrale Instanz kann ihre Macht ausnutzen, um Daten zu manipulieren oder zu zensieren.

Dies hat natürlich auch Schattenseiten. So fanden Wissenschaftler der Universität Aachen jüngst heraus, dass in den Tiefen der Bitcoin-Blockchain auch Links zu kinderpornografischen Inhalten im Darkweb versteckt sind (Matzutt, 2018). Auch solche eindeutig kriminellen und verwerflichen Inhalte können nachträglich nicht mehr aus der Blockchain gelöscht werden. Allerdings wäre es wohl eher Zufall, über solche Darkweb-Links zu stolpern, da diese einerseits codiert sind und erst übersetzt werden müssten und andererseits in der großen Anzahl der Blöcke untergehen.

Geheime Nachrichten in der Blockchain

Jeder Block in der Bitcoin-Blockchain besitzt ein Feld, das *Coinbase* genannt wird. Dieses Feld ist eine Zeichenkette im Hexadezimalsystem, die vom erfolgreichen Miner des jeweiligen Blocks beliebig gewählt werden kann. Schnell wurde es unter Minern beliebt, Zeichenketten zu hinterlassen, die übersetzt ins ASCII-Format versteckte Botschaften darstellen. Schon im ersten Block der Bitcoin-Blockchain hat Satoshi Nakamoto folgende Nachricht hinterlassen: »The Times 03/Jan/2009 Chancellor on brink of second bailout for banks.« Die Nachricht enthält die Überschrift der Titelseite der Zeitung »The Times« an dem Tag, an dem die Bitcoin-Blockchain in Betrieb genommen wurde. Mit ihr wollte Nakamoto beweisen, dass der Block wirklich erst an diesem Tag erzeugt wurde. Neben der Coinbase gibt es noch mehr Möglichkeiten, verschlüsselte Nachrichten zu hinterlassen, zum Beispiel durch die Wahl der Netzwerkadresse. In der Blockchain finden sich auf diese Weise viele kuriose Nachrichten, wie Heiratsanträge, Zitate, Bilder oder auch eine Datei von Wikileaks.

Die Unveränderbarkeit der Blockchain führt aber auch zu Vertrauen zwischen Parteien. Keine einzelne TTP, sondern das gesamte Netzwerk sorgt dafür, dass Transaktionen wirklich ausgeführt und Vereinbarungen eingehalten werden. Diese sichere Einhaltung ermöglicht auch digitale Verträge, sogenannte Smart Contracts, die in Kapitel 9 vorgestellt werden. Auch die Transparenz der Blockchain, in der alle Informationen für jeden zugänglich sind, schafft Vertrauen für die Anwender. So können Sie sich sicher sein, dass Ihr Vertragspartner wirklich genug Geld hat oder auch wirklich im Besitz einer Sache ist.

Das Double-Spending-Problem wird durch die Blockchain ebenfalls erfolgreich gelöst. Dabei wird nicht der Versuch des Double-Spending an sich verboten. Wenn Sie versuchen würden, Ihren Bitcoin zwei Personen gleichzeitig zu überweisen, könnten Sie das natürlich probieren. Das Blockchain-System würde es allerdings verhindern, da jede Transaktion durch das Netzwerk, bestehend aus einzelnen Knoten, mehrfach verifiziert und bestätigt werden muss. Der Betrugsversuch würde hierbei auffallen. Dennoch gibt es theoretische Szenarien, mit denen ein Double-Spending erfolgreich durchführbar wäre. Diese stellen wir Ihnen in Abschnitt 3.5 vor.

2.2 Geschichte der Blockchain

Natürlich gibt es nicht die universelle Blockchain, und Satoshi Nakamotos ursprüngliche Ideen wurden in zahlreichen neuen Projekten verändert und ausgebaut. So entwickelten Projekte wie *Monero* oder *ZCash* Kryptowährungen, bei denen Transaktionen nicht transparent gespeichert werden müssen und daher nicht immer für jedermann einsehbar sind. Projekte wie *Ethereum* entwickelten ganze Plattformen, bei denen es nicht nur möglich ist, Zahlungsvorgänge abzubilden, sondern auch Programmiercode in der Blockchain zu speichern. So entstanden im Laufe der Zeit viele unterschiedliche Blockchains, die eigene Netzwerke gebildet haben. Wie sich diese Weiterentwicklung vollzogen hat, möchten wir Ihnen in diesem Abschnitt vorstellen.

2.2.1 Pioniere der Blockchain

Häufig wird der Anfang der Blockchain-Technologie mit dem Start von Bitcoin gleichgesetzt. Es gab allerdings schon viel früher erste Überlegungen zu dieser neuen Technologie. 1983 veröffentlichte David Chaum seine Publikation »Blind Signatures for Untraceable Payments«, in der er ein auf Kryptografie aufbauendes Konzept vorstellte, um anonyme Zahlungen zu verwirklichen. Sein Wissen setzte er 1995 schließlich in seinem Projekt *eCash* um, das als eines der ersten Systeme digitales Geld ermöglichte, um Mikrotransaktionen als Kleinstüberweisungen zu ermöglichen. Neben eCash gab es noch weitere Projekte, die allerdings nach kurzer Zeit wieder eingestampft wurden.

1997 beschrieb Nick Szabo in einer Publikation das Konzept für Smart Contracts. Im gleichen Jahr stellte Adam Back das sogenannte *Hashcash* vor, das zum ersten Mal Rechenleistung zur Wertschöpfung nutzte. Der Sender einer E-Mail sollte sich durch die Aufwendung von eigener Rechenleistung ein virtuelles Porto, das Hashcash, generieren. Auf diesem Weg sollte Spam unterbunden werden, da der Aufwand für den Versender dieser millionenfachen Nachrichten schlicht zu teuer werden würde. Adam Back berücksichtigte in seiner Idee das *Proof-of-Work*-Verfahren (*PoW*), das auch heute viele Kryptowährungen zur Konsensfindung nutzen und dass wir Ihnen in Abschnitt 3.2.3 vorstellen.

Dieses Verfahren wurde 1998 von Wei Dai in seinem Projekt b-money (*http://www.weidai.com/bmoney.txt*) weiterentwickelt. Das Konzept sah bereits einen PoW zur Generierung der projekteigenen Währung und die Möglichkeiten zum Eingehen von Verträgen vor. Ebenfalls in 1998 veröffentlichte Nick Szabo seine Idee zu *Bit Gold*, das ebenfalls PoW zur Konsensfindung und zur Generierung der Einheiten von Bit Gold nutzte. Szabo wollte ein Konzept erschaffen, das das Double-Spending-Problem ohne zentrale Instanz löst. Dabei nahm er sich echtes Gold als Vorbild.

2005 verknüpfte Hal Finney die Ideen von Adam Backs Hashcash und Wei Dais b-money und stellte sein Konzept *Reusable Proof-of-Work* vor, das den Austausch von digitalem Geld im Netzwerk vereinfachte. Allerdings sah dieses Konzept immer noch eine zentrale Einheit zur Zuordnung der Geldeinheiten zu Nutzern vor. Kaum eine dieser Ideen der Pioniere der Blockchain-Technologie, mit der Ausnahme von eCash, wurde tatsächlich umgesetzt. Es war also an der Zeit, ein ausgereiftes Konzept zu entwickeln, das sämtliche Zentralität beseitigt und das konsequent umgesetzt wird.

2.2.2 Bitcoin

2007 war kein gutes Jahr für die Finanzbranche. Eine Krise, die am US-Immobilienmarkt begann, breitete sich im Sommer 2007 auf das gesamte Finanzsystem aus und beeinträchtigte die Weltwirtschaft. Die Krise wirkte sich massiv auf den Aktienmarkt, Zinsen und auch auf ganze Volkswirtschaften aus. Haben Sie damals mit Aktien gehandelt? Dann haben Sie die Auswirkungen sicherlich am eigenen Leib erfahren. Die Krise ging mit einem massiven Imageverlust für die gesamte Finanzbranche einher, und viele Menschen verloren ihr Vertrauen in Banken. Im September 2008 erreichte die Krise mit der Pleite der Bank Lehman Brothers ihren Höhepunkt und zog einen Börsencrash nach sich. Nur einige Wochen später, im November 2008, veröffentliche Satoshi Nakamoto sein Whitepaper »Bitcoin: A Peer-to-Peer Electronic Cash System«, das eine Währung beschreibt, die komplett ohne Banken und staatliche Institutionen auskommt und das Potenzial hat, das bisherige Finanzsystem zu revolutionieren.

Bitcoin-Fakten

Als erste Kryptowährung ist Bitcoin zugleich auch die beliebteste. Die offizielle Abkürzung, die auch an den Handelsplätzen verwendet wird, ist BTC. Die kleinste Einheit von Bitcoin nennt sich *Satoshi*, wobei ein Satoshi 0,00000001 Bitcoin entspricht. Die endgültige Menge an Bitcoins ist rechnerisch begrenzt und beträgt insgesamt 21.000.000 Bitcoin. Es ist im Bitcoin-Protokoll festgelegt, dass diese finale Anzahl im Jahr 2140 erreicht werden wird.

Aktuell (Stand Dezember 2018) werden pro Tag 1.800 Bitcoin gemined. Pro Block werden aktuell 12,5 Bitcoin als Belohnung an die Miner ausgeschüttet, was bedeutet, dass insgesamt am Tag 144 Blöcke zur Bitcoin-Blockchain hinzugefügt werden. Am Anfang des Projekts Bitcoin lag die Anzahl der täglich ausgeschütteten Bitcoins noch bei 7.200. Diese Anzahl halbiert sich allerdings alle vier Jahre, um durch Verknappung eine inflationäre Entwicklung der Coins zu vermeiden. Im Jahr 2020 wird das nächste sogenannte *Halving* stattfinden.

Tagesaktuelle Informationen rund um Bitcoin finden Sie auf der Homepage unter *https://blockchain.info/stats*.

Das Bitcoin-Whitepaper wurde von Satoshi Nakamoto über eine Mailingliste für Kryptografie verschickt und beschreibt das Konzept der neuen Kryptowährung. Nakamoto lehnte seine Erfindung dabei nach eigener Aussage an Wei Dais b-money und Nick Szabos Bit Gold an. Mit seinem Konzept stieß Nakamoto aber auch auf viele Zweifler. In seinen ersten E-Mails versuchte er, Einwände wie »Das System skaliert nicht« oder die Gefahr der Übernahme des Netzwerks durch Zombiefarmen zu entkräften. Den gesamten E-Mail-Verkehr der Bitcoin-Geburtsstunde finden Sie unter *https://satoshi.nakamotoinstitute.org/emails*.

Um zu beweisen, dass sein Konzept funktioniert, nahm er das Bitcoin-Netzwerk am 3. Januar 2009 in Betrieb. An diesem Datum wurde der sogenannte *Genesis Block*, der Block #0 der Bitcoin-Blockchain, und mit ihm die ersten 50 Bitcoin erschaffen. Die Gutschrift dieser Bitcoins auf das Wallet von Nakamoto war übrigens die erste Transaktion im Bitcoin-Netzwerk. Diese ersten Bitcoins konnten aufgrund einer Eigenheit im Quellcode von Bitcoin nie transferiert werden. Ob dies ein Fehler, eine Nachlässigkeit oder Absicht von Nakamoto war, ist nicht bekannt. Wie bereits erwähnt, befindet sich im ersten Block die damals tagesaktuelle Überschrift der Titelseite der britischen Zeitung »The Times«, die aussagt, dass der damalige Schatzkanzler Alistair Darling ein zweites Rettungspaket für Banken in Aussicht stellt. Es wird häufig spekuliert, ob Nakamoto absichtlich diese Überschrift gewählt hat, die auf die globale Finanzkrise referenziert.

Am 9. Januar 2009 veröffentlichte Nakamoto die Software *Bitcoin Core*, um anderen Nutzern den Zugriff auf das Netzwerk zu ermöglichen. Die erste Transaktion zu

einem anderen Teilnehmer des Bitcoin-Netzwerks erfolgte schließlich am 12. Januar 2009 und wurde in Block #170 festgehalten. Dem aufmerksamen Leser dürfte der Name des Empfängers bereits ein Begriff sein: Krypto-Pionier Hal Finney, der Erfinder des Konzepts *Reusable Proof-of-Work*, bekam 10 Bitcoins von Satoshi Nakamoto transferiert. Finney ist daher in der Szene ein heißer Kandidat, wenn es um die wahre Identität von Satoshi Nakamoto geht. Leider ist er 2014 verstorben und hat, falls er wirklich der Erfinder von Bitcoin war, sein Geheimnis mit ins Grab genommen.

Die neue virtuelle Währung machte vor allem in der Computer- und Technikszene schnell von sich reden. Am 5. Oktober 2009 veröffentlichte die Zeitung New Liberty Standard einen ersten Umrechnungskurs für Bitcoin, der sich an dem Kostenaufwand orientiert, der notwendig ist, um einen Bitcoin zu produzieren. Ein US-Dollar entsprach dabei 1.309,03 Bitcoin. Dazu sei gesagt, dass Bitcoins damals noch nicht auf klassischen Börsen gehandelt werden konnten. Bitcoin-Käufe und -Verkäufe wurden in Foren oder über Dienste wie *LocalBitcoins* abgewickelt. Bei LocalBitcoins konnten Verkäufer Inserate schalten und sich dann mit Kaufinteressenten treffen, um Bitcoins zu verkaufen.

Der Handel mit der Kryptowährung wurde deutlich vereinfacht, als im Juli 2010 *MtGox* als damals erste Börse für Bitcoins seine Pforten öffnete. MtGox gab es schon länger, allerdings wurden bis dahin lediglich Karten des Spiels »Magic: The Gathering« auf der Plattform getauscht. Im gleichen Monat stieg der Bitcoin-Kurs das erste Mal über 0,08 US-Dollar. Am Ende des gleichen Jahres explodierte der Bitcoin-Kurs plötzlich und stieg auf über 30 Dollar. Als er im Jahr darauf wieder in den einstelligen Bereich fiel, wurde schon das Ende der Währung vorausgesagt.

Die Bitcoin-Pizza

Die Geschichte über den ersten dokumentierten Einkauf mit Bitcoins gehört zu den Klassikern in der Szene. Daher wollen wir diese Anekdote natürlich auch Ihnen nicht vorenthalten. Laszlo Hanyecz, ein Entwickler, der dabei half, das Projekt Bitcoin weiterzuentwickeln, fragte 2010 in einem Forum, wer ihm zwei Pizzen für 10.000 Bitcoin verkaufen würde. Dies entsprach damals ungefähr 30 Euro.

Am 22. Mai 2010, drei Tage nach seinem Angebot, fand sich endlich ein williger Handelspartner, und der Deal konnte über die Bühne gehen. Im Nachhinein war es kein schlechtes Geschäft für den Verkäufer, wenn Sie bedenken, dass diese 10.000 Bitcoin im Dezember 2017 rund 160 Millionen Euro wert waren. Jedes Jahr am 22. Mai wird in der Community noch heute der *Bitcoin Pizza Day* gefeiert, um diesem besonderen Ereignis zu gedenken.

Im Jahr 2011 öffnete mit *Silk Road* ein Marktplatz seine Pforten, der lediglich Bitcoin als Zahlungsmittel akzeptierte. In Kombination mit der Verwendung des *Tor-Netzwerks* war es dort für Anwender möglich, anonym einzukaufen. Da auf der Plattform

im Darknet hauptsächlich mit Drogen gehandelt wurde, schloss im Jahr 2013 das FBI den Marktplatz wieder. Der Ruf der »Währung für Kriminelle« haftet Bitcoin allerdings noch heute an.

Ende des Jahres 2013 stieg der Bitcoin in schwindelerregende Höhen und wies im November 2013 einen Kurs von 1.242 Dollar auf. Zwar erreichte der Kurs an dieser Stelle vorerst seinen Höhepunkt, behielt aber ein hohes Niveau. Der Höhenflug fand sein vorläufiges Ende, als MtGox im Februar 2014 Insolvenz anmeldete und erklärte, dass insgesamt 850.000 Bitcoin gestohlen worden waren. Bereits kurz vorher hatte die Plattform alle Transaktionen gestoppt, sodass es den Kunden nicht mehr möglich war, Abhebungen zu tätigen. Der Besitzer von MtGox behauptete zuerst, dass die Bitcoins von externen Hackern gestohlen worden waren, Ermittlungen ergaben jedoch schnell, dass ein Großteil des Diebstahls intern getätigt wurde. Der Bitcoin-Kurs brauchte einige Jahre, um sich von diesem Schlag zu erholen.

Im Laufe der Zeit kristallisierten sich einige Probleme der Bitcoin-Implementierung heraus. Die Skalierung des Netzwerks entwickelte sich bald zur Achillesferse des Netzwerks. Bitcoin kann circa sieben Transaktionen pro Sekunde abarbeiten, während der Kreditkartenservice Visa auf rund 24.000 Transaktionen pro Sekunde kommt. Als Grund hierfür wird häufig die limitierte Blockgröße von 1 MByte diskutiert. Diese Limitierung wurde von Nakamoto im Sourcecode implementiert und kann nur durch erheblichen Aufwand geändert werden (*Hard Fork*). In Kombination mit der Tatsache, dass durchschnittlich nur alle zehn Minuten ein neuer Block entsteht, führt dies zu einer begrenzten Transaktionsanzahl, die vom Netzwerk verarbeitet werden kann. Daher werden sowohl eine Erhöhung der Blockgröße als auch eine Verkürzung der Entstehungszeit für neue Blöcke diskutiert, wobei beide Ansätze auch Nachteile mit sich bringen. Durch die Erhöhung der Blockgröße könnte das Mining an Profitabilität einbüßen, was zu weniger Minern führen könnte. Unter einer Senkung der Entstehungszeit für neue Blöcke würde die Sicherheit von Bitcoin leiden. Um das Problem zu lösen, wurde im Juli 2017 das Upgrade *Segregated Witnesses* (*SegWit*) durchgeführt, das den Aufbau einer Transaktion so ändert, dass sie weniger Platz in Anspruch nimmt und so letztendlich mehr Transaktionen in einen Block passen. Da hierfür die Struktur des Blocks nicht geändert werden musste, reichte ein *Soft Fork*, um dies zu realisieren. Der zweite Teil des Updates, *SegWit2x* genannt, sollte im Herbst 2017 folgen und die Blockgröße auf 2 MByte erhöhen. Dies wäre nur durch einen Hard Fork möglich gewesen, der von der Entwicklungscommunity allerdings in letzter Minute abgesagt wurde.

Forks

Der Begriff *Fork* kann im Deutschen mit Verzweigung übersetzt werden und beschreibt in der Softwareentwicklung die parallele Weiterentwicklung eines Softwareprojekts in einem separaten Zweig. Zu einem Zeitpunkt im Laufe des Projekts

wird der aktuelle Stand kopiert und unabhängig vom Mutterzweig weitergeführt. Der alte und der neue Zweig behalten dann eine gemeinsame Vergangenheit, die bis zur Verzweigung identisch ist, entwickeln sich aber danach unterschiedlich weiter.

Das Prinzip der Forks hat sich auch für Blockchains durchgesetzt, wodurch beispielsweise größere Änderungen der Software durchgeführt werden können. Die Teilnehmer im Netzwerk müssen sich allerdings entscheiden, ob sie die alte oder die neue Software verwenden möchten, und somit, welchen Zweig sie unterstützen. Grundsätzlich wird zwischen zwei Arten von Forks unterschieden: *Soft Forks* und *Hard Forks*.

Bei einem Soft Fork ist die neue Version der Software abwärtskompatibel, und Knoten mit alter Software können mit Knoten mit neuer Software zusammenarbeiten. Dies wird häufig bei kleineren Änderungen in der Software einer Blockchain gemacht. Alte Knoten akzeptieren bei einem Soft Fork Blöcke der neuen Version, allerdings akzeptieren die neuen Knoten keine Blöcke der alten Version. Wenn im Laufe der Zeit immer mehr alte Blöcke abgelehnt werden, motiviert dies die Teilnehmer ebenfalls, auf die neue Version umzusteigen. Hierbei verzweigt sich die Blockchain als Datenstruktur nicht, sondern es bleibt bei einem Zweig, der von beiden Versionen gemeinsam fortgesetzt wird.

Bei einem Hard Fork können Knoten mit neuer Version nicht mehr mit den Knoten der alten Version zusammenarbeiten. Das neue Protokoll lehnt generierte Blöcke der alten Version und die alte Software lehnt Blöcke der neuen Version ab. Es entstehen also zwei verschiedene Zweige der Blockchain, einer mit Blöcken der neuen Version und einer mit Blöcken der alten Version. Diese Zweige werden auch als Blockchain Forks bezeichnet. Wenn sich die Knoten im Netzwerk nicht auf eine Version und somit einen Zweig einigen, ist das die Geburtsstunde einer neuen Blockchain und gegebenenfalls Kryptowährung, die sich fortan unabhängig vom Mutterzweig entwickelt. Dies wird dann als *Fork gone wrong* bezeichnet, wobei es aber durchaus Projekte gibt, die die Teilung einer Blockchain bewusst erreichen möchten.

Ein weiteres Problem von Bitcoin stellt die zunehmende Zentralisierung des Netzwerks dar, das eigentlich genau mit der gegenteiligen Dezentralisierung wirbt. Die Verbreitung von *ASIC-Minern* (ASIC steht für *Application-Specific Integrated Circuit*) hat dazu geführt, dass das klassische Mining mit dem heimischen PC nicht mehr lukrativ ist. Bei ASICs handelt es sich um integrierte Schaltkreise, die für einen bestimmten Zweck gestaltet sind. Die Funktionalität wird vom Hersteller bei der Herstellung vorgegeben und kann vom Anwender nicht mehr verändert werden. Damit unterscheiden sie sich von den CPUs (*Central Processing Units*), die in unseren Computern als Prozessor dienen, oder von den GPUs (*Graphical Processing Units*) auf Grafikkarten. Unsere Grafikkarte ist vielseitig einsetzbar. Wir können mit ihr Mining betreiben, aber auch Spiele spielen oder Videos anschauen. ASIC-Miner sind speziell für das

Mining einer bestimmten Kryptowährung ausgelegt und eignen sich für keine andere Nutzung. Oft werden diese Miner gewerblich genutzt, was einen großen Teil der Rechenleistung in wenige zentrale Knoten oder Mining-Farmen legt. Außerdem wurden in der Vergangenheit ASIC-Miner von Herstellern mit Software ausgestattet, die so manipulierbar war, dass die Hersteller von außen auf den Miner zugreifen konnten. Dieser Eingriff bietet einen Angriffspunkt für das Bitcoin-Netzwerk.

2.2.3 Altcoins

Auch wenn Bitcoin die erste Kryptowährung auf Blockchain-Basis war, sollte sie nicht lange der einzige Akteur in diesem neuen Umfeld bleiben. Kurz nach der Entstehung von Bitcoin entstanden die ersten alternativen Blockchain-Projekte, genannt *Altcoins*, die ihre eigene Implementierung einer Blockchain nutzen. Mit *Namecoin* ging bereits im April 2011 der erste Altcoin ins Rennen. Namecoin wurde entwickelt, um ein unabhängiges *Domain Name System* (DNS) auf Blockchain-Basis zu erschaffen. Die Altcoins, die danach entstanden, versuchten sich allerdings eher am Anwendungsfall von Bitcoin und stellten digitales Geld dar. Im Oktober 2011 erblickte *Litecoin* das Licht der Welt. Die von Charlie Lee entwickelte Kryptowährung änderte die Implementierung von Bitcoin leicht ab. So verwendet Litecoin einen anderen Hashalgorithmus und generiert schneller Blöcke.

Die Anzahl der Altcoins stieg über die Jahre stark an, und dabei tauchten auch einige kuriose Währungen auf. 2013 entstand der *Dogecoin*, der ursprünglich als Parodie für Kryptowährungen gedacht war, auf Basis der Litecoin-Implementierung. Dogecoin ist nach dem Internetphänomen *Doge* benannt und hat sich zu einer der wertvollsten Kryptowährungen entwickelt. Eine andere Kuriosität stellt der Altcoin *Coinye* dar, der auf den amerikanischen Musiker Kanye West anspielt und dessen karikiertes Konterfei als Logo nutzt. Der langfristige Erfolg dieser Währung blieb allerdings aus.

Die bisher beschriebenen Altcoins ließen sich zwar von Bitcoin inspirieren, starteten aber als unabhängige Projekte und etablierten ihre eigene Blockchain. Einige der Altcoin-Projekte sahen sich allerdings als direkte Nachfolger von Bitcoin und bauten auf der bisherigen Bitcoin-Blockchain mit all ihren Transaktionen auf. Dies wurde mit Hard Forks bewerkstelligt. Wie in Abbildung 2.2 zu sehen, machte im August 2017 die Währung Bitcoin Cash den Anfang. *Bitcoin Cash* (BCH) will die Skalierungsproblematik von Bitcoin beseitigen, indem die Blockgröße auf 8 MByte angehoben wurde. Der Hard Fork, der zu Bitcoin Gold führte, fand im Oktober 2017 statt. *Bitcoin Gold* hat das Anliegen, die Zentralisierungsproblematik des Bitcoin-Netzwerks zu lösen, indem ein neuer Hashalgorithmus genutzt wird, der gegen ASIC-Miner resistent ist. Im Februar 2018 spaltete sich *Bitcoin Private* ab. Die Kryptowährung möchte verhindern, dass die Adressen der an einer Transaktion beteiligten Parteien transparent in der Blockchain abgebildet werden. Ist die Identität hinter einer Adresse bekannt, können

sämtliche Kontenbewegungen oder das Guthaben nachvollzogen werden. Bitcoin Private löst dieses Problem, indem Absender und Empfänger einer Transaktion in der Blockchain für die Öffentlichkeit unleserlich hinterlegt werden.

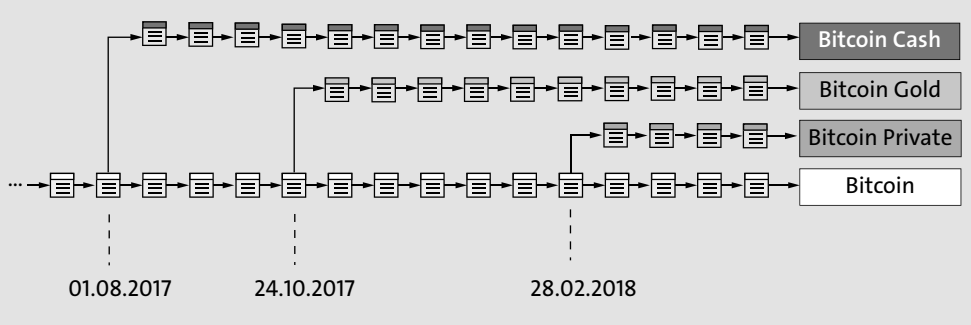

Abbildung 2.2 Die verschiedenen Hard Forks der Bitcoin-Blockchain

Insgesamt gibt es aktuell über 1.600 verschiedene Altcoins (Stand Juni 2018) mit steigender Tendenz. Diese haben sich im Laufe der Jahre der sich stetig weiterentwickelnden Blockchain-Technologie angepasst.

Während Altcoins zu Beginn größtenteils klassische Kryptowährungen darstellten, entstanden schon bald Projekte, die mehr mit der Blockchain vorhatten, als nur Währungen zu erschaffen. Nachdem Namecoin bereits sehr früh den Anfang gemacht hatte, entstanden 2013 die sogenannten *Colored Coins*. Das Projekt hatte die Idee, Nutzern der Bitcoin-Blockchain zu ermöglichen, einzelne Bitcoins mit Sachwerten zu verknüpfen, indem Bitcoins Farben zugeordnet wurden. Anwender könnten beispielsweise mehrere Bitcoins hellgrün färben und diese zukünftig als digitale Repräsentation für den Besitz eines Hauses verwenden. Realisiert wurde Colored Coins über die Speicherung von Metadaten in der Blockchain.

Ein Ansatz zur Erweiterung der Bitcoin-Blockchain sind auch die sogenannten Metacoins. Diese besitzen ein Protokoll, das auf Bitcoin aufsetzt und Transaktionen des Metacoins in Bitcoin-Transaktionen verpackt. Diese Ansätze wurden allerdings nie weitreichend adaptiert. Der Sprung zur Transaktions- und Anwendungsplattform, der sogenannten Blockchain 2.0, schaffte erst das Projekt *Ethereum*, das wir Ihnen im nächsten Abschnitt genauer vorstellen möchten.

2.2.4 Blockchain 2.0

Der Programmierer Vitalik Buterin hat es in der Blockchain-Szene zu einer ähnlichen Berühmtheit gebracht wie Satoshi Nakamoto. Kein Wunder, schließlich hat er einen ähnlich wertvollen Beitrag zu Weiterentwicklung der Technologie geleistet: die Etablierung der *Blockchain 2.0*.

Im Jahr 2013 stellte er sein Projekt Ethereum erstmals der Blockchain-Community in einem Whitepaper vor. Dabei war auch sein Konzept nicht gänzlich neu, sondern griff, wie schon vorher Bitcoin, Ideen der bereits vorgestellten Blockchain-Pioniere auf, um diese in dem neuen Ökosystem weiterzuentwickeln. So hatte vor allem Nick Szabo mit seinen Arbeiten zu Smart Contracts oder der Verwaltung von Grundstücken in einer dezentral verteilten Datenbank bereits Ideen vorgestellt, die Buterin in seinem Projekt betrachtete.

Buterin hatte die Idee, mit Ethereum eine Art Plattform basierend auf der Blockchain-Technologie zu entwickeln, auf der es möglich sein sollte, dezentral verteilte Applikationen laufen zu lassen. Dies stellte eine Weiterentwicklung der bisherigen Blockchains dar, die in den meisten Fällen versuchten, Geld zu simulieren. Auch die Projekte Namecoin, Colored Coins oder die Metacoins erweiterten die Funktionalität der Blockchain, hatten aber jeweils einen eingeschränkten Anwendungszweck. Buterin wollte all diese Konzepte zusammenführen und verbessern, um die Blockchain programmierbar und universell nutzbar zu machen.

Vitalik Buterin

Vitalik Buterin wurde 1994 in Russland geboren und zog bereits als Kind mit seinen Eltern nach Kanada. Als Sohn eines Informatikers zeigte sich auch bei Buterin bereits in jungen Jahren die Neigung zu Computern und Mathematik. Durch seinen Vater wurde er auch an die Bitcoin-Technologie herangeführt. Bald schrieb er Artikel für Blogs und auch die Zeitschrift *Bitcoin Magazine* über das virtuelle Geld. 2013 stellte er in einem Whitepaper das Projekt Ethereum erstmals der Öffentlichkeit vor. 2014 erhielt Buterin 100.000 Dollar von der Thiel Foundation des PayPal-Mitbegründers Peter Thiel. Das Geld ermöglichte Buterin, sein Studium abzubrechen, um sich in Vollzeit um Ethereum zu kümmern.

Buterin legte sein Hauptaugenmerk auf die Smart Contracts, also digitale Verträge, die in der Blockchain abgebildet werden sollen. Buterin bemerkte in seinem Whitepaper, dass mit Bitcoin bereits erste wenn auch schwache Umsetzungen solcher Smart Contracts realisierbar seien. Jedoch habe die Entwicklung mit der Skriptsprache von Bitcoin einige entscheidende Einschränkungen, wie die fehlende Turing-Vollständigkeit oder die limitierten Zustände. Für die Transaktion von Bitcoins sind diese Features nicht notwendig. Außerdem werden so böswillige Hacker davon abgehalten, mit komplizierten Transaktionen die Rechenleistung des Netzwerks in die Knie zu zwingen, das ja lediglich auf den Austausch von Coins ausgelegt ist. Diese Einschränkungen löst Ethereum bewusst auf, um auch anspruchsvollere Anwendungsfälle möglich zu machen.

Die Smart Contracts von Buterin werden im Whitepaper als kryptografische Boxen beschrieben, die einen Wert enthalten, der nur freigeschaltet wird, wenn eine oder

mehrere Bedingungen erfüllt werden. Durch die Anwendung dieser Smart Contracts macht Ethereum es möglich, Anwendungen zu entwickeln, die nicht unterbrochen oder zensiert werden können oder von Betrug bedroht sind. Vielmehr kümmert sich das Blockchain-System um die Durchsetzung der Vertragsbedingungen. Zur Programmierung dieser Anwendungen ist die eigens konzipierte Programmiersprache *Solidity* vorgesehen. Das Zahlungsmittel und die Währung im Netzwerk wird *Ether* genannt.

Ether-Fakten

Ether ist der »Treibstoff« des Ethereum-Netzwerks und ähnlich wie reguläre Kryptowährungen. Mit Ether – bei den gängigen Börsen für Kryptowährungen abgekürzt ETH – kann der Betrieb von dezentralisierten Applikationen, genannt DApps, oder Smart Contracts im Netzwerk gezahlt werden. Bei der Wahl des Namens hat sich Vitalik Buterin von Aristoteles und seiner Theorie vom alles durchsetzenden Ether inspirieren lassen.

Wie Bitcoin hat auch Ether kleinere Einheiten, in die es sich aufteilen lässt. Diese sind nach Pionieren der Kryptowährungen benannt. Ether lässt sich allerdings in ganze 18 Nachkommastellen unterteilen. Die kleinste Einheit nennt sich Wei. Ein Wei ist 0,000000000000000001 Ether wert. Weitere Einheiten sind zudem Szabo (0,000001 Ether) und Finney (0,001 Ether).

Im Ethereum-Netzwerk wird durchschnittlich alle 15 Sekunden ein neuer Block erzeugt, für den jeweils 3 Ether an den Miner ausgeschüttet werden. Der Genesis Block der Ethereum-Blockchain schüttete 60.102.216 Einheiten aus. Das war die Menge, die Ethereum schon vor dem Start des Projekts an Interessenten verkauft hatte. Die Anzahl der neuen Ether, die jährlich ausgeschüttet werden, beträgt 15.626.576 und bleibt von Jahr zu Jahr gleich. Dies sorgt zwar für eine stetige Inflation, die sich allerdings prozentual aufgrund der wachsenden Anzahl der Ether immer mehr verringert.

Tagesaktuelle Informationen zur Ethereum-Blockchain können Sie auf der Homepage *https://etherscan.io/* einsehen.

Ein weiteres vielversprechendes Konzept, das Buterin bereits in seinem Whitepaper vorstellte, sind die *Dezentralisierten Autonomen Organisationen* (*DAO*). Diese virtuellen Organisationen werden durch mehrere Smart Contracts definiert, die sich um die Abwicklung bestimmter Aufgaben kümmern. Sie können sich eine solche DAO als autonomen Verein vorstellen. Einzelne Mitglieder können Anträge stellen und andere Mitglieder darüber transparent abstimmen. Die automatisierte Mitgliedschaftsverlängerung durch die Zahlung des Beitrags kann genauso abgebildet werden wie Überschussausschüttungen an Mitglieder – alles programmiert und festgelegt in Smart Contracts.

Ein weiterer klassischer Anwendungsfall sind die bereits erwähnten DApps. Hierbei handelt es sich um Anwendungen, die nicht auf einem einzelnen Computer oder Server laufen, sondern in einem Netzwerk. Dabei gab es DApps auch schon in Peer-to-Peer-Netzwerken (P2P-Netzwerken) vor der Blockchain. Bei Ethereum nutzen die Applikationen Smart Contracts. DApps werden in Kapitel 16 genauer betrachtet.

Kurze Zeit nach dem Release des Whitepapers folgte das Yellow Paper »Ethereum: A Secure Decentralised Generalised Transaction Ledger« von Dr. Gavin Wood. Er arbeitete eng mit Buterin und dem Projekt zusammen. In dem Yellow Paper spezifizierte er Ethereum und auch die *Ethereum Virtual Machine* (*EVM*).

Im Juli und August 2014 begann schließlich der Crowdsale von Ethereum, bei dem eine gewisse Anzahl von Ether bereits vorab zum Verkauf angeboten wurde. Solche Crowdsales werden in der Szene *Initial Coin Offerings* (*ICO*) oder auch *Initial Token Sale* (*ITS*) genannt. Auf diese gehen wir in Abschnitt 2.3.4 genauer ein.

Der Vorverkauf war ein voller Erfolg. Vitalik Buterin und sein Team konnten 18,4 Millionen Dollar einsammeln und so die Entwicklung von Ethereum sicherstellen. Diese erfolgte in der gemeinnützigen Ethereum Foundation, die kurz vor dem Crowdsale gegründet wurde und in der Schweiz beheimatet ist.

Im Mai 2015 wurde die erste Testversion von Ethereum online gestellt. Zwei Monate später folgte die erste Entwicklungsstufe der Plattform. Auch wenn Ethereum von diesem Moment an nutzbar war, befindet sich die Plattform auch aktuell noch in der Entwicklungsphase und wird vermutlich erst in den kommenden Jahren fertiggestellt.

Die Ethereum-Roadmap

Ethereum ist in vier Entwicklungsphasen aufgeteilt, wobei die Funktionalität der Plattform mit jeder Phase erweitert wird. In Ergänzung zu den offiziellen Phasen wurden auch einige Hard Forks durchgeführt, um ein paar Schwachstellen des Netzwerks schnell zu schließen. Im Folgenden werden die einzelnen Phasen des Projekts vorgestellt.

Die erste Phase, *Frontier*, wurde im Juli 2015 veröffentlicht. Sie bot noch relativ wenig Funktionsumfang und bestand lediglich aus der grundlegenden dezentral verteilten Plattform. Dennoch konnten Nutzer bereits Ether handeln und transferieren sowie DApps und Smart Contracts entwickeln. Zudem stellte die Phase noch keine Ansprüche an die Sicherheit, was auch in roter Warnschrift auf der offiziellen Ethereum-Homepage betont wurde.

Die zweite Phase, *Homestead*, wurde im März 2016 veröffentlicht. Homestead brachte vor allem Änderungen am Protokoll mit sich und wurde nun als sicher eingestuft. Bei Homestead handelte es sich um die erste stabile Version von Ethereum.

Die dritte Phase, *Metropolis*, unterteilt sich in die zwei Stufen *Metropolis: Byzantium* und *Metropolis: Constantinople*. Byzantium wurde im Oktober 2017 veröffentlicht und verbesserte unter anderem die Privatsphäre der Blockchain und der Sicherheit der Smart Contracts. Zudem wurde die sogenannte *Difficulty Timebomb* zurückgesetzt, was den Entwicklern mehr Zeit gibt, die Plattform zu entwickeln. Die Zeitbombe wurde ursprünglich eingebaut, um den Wechsel vom rechenintensiven PoW zu dem ressourcenschonenden *Proof-of-Stake-Verfahren* (*PoS*) zu unterstützen. Da sich das jedoch als schwierig herausstellte, wurde diese Frist mit Byzantium verlängert. Der Übertritt in die Stufe Constantinople wird für das Frühjahr 2019 erwartet. Hier werden vor allem kleine Verbesserungen durchgeführt und Vorbereitungen für die letzte Phase getroffen.

Die vierte und letzte Phase, *Serenity*, soll einige große Änderungen mit sich bringen. Eine dieser Änderungen ist die Umstellung auf PoS. Serenity wird auch als Ethereum 2.0 bezeichnet. Mehr zu den einzelnen Phasen erfahren Sie in Abschnitt 3.3.7.

Durch die Etablierung von Ethereum wurden auch die ICOs zunehmend populärer. Dies hing vor allem mit der Tatsache zusammen, dass es durch die Plattform neuen Projekten wesentlich erleichtert wurde, eigene Projekte zu entwickeln. Durch einen standardisierten Smart Contract konnte relativ einfach ein eigener Coin oder ein eigenes Token erstellt werden (*ERC-20 Token Standard*). Mehr zu diesen Standards wird in Kapitel 15 erläutert. Der erste ICO auf Ethereum war das Projekt *Augur*, das im August 2015 stattfand, also kurz nach der Veröffentlichung der ersten Entwicklungsstufe. Der ICO sammelte Ether im Wert von 5 Millionen Dollar ein.

Einige Monate später, im April 2016, sollte ein auf Ethereum stattfindender ICO jedoch sämtliche Rekorde brechen. Das Projekt *The DAO* sammelte in seinem Crowdsale Ether mit einem Wert von über 150 Millionen Dollar ein. The DAO hatte sich vorgenommen, erstmals die von Buterin im Whitepaper vorgestellten *Dezentralisierten Autonomen Organisationen* umzusetzen, und sollte ein Vorzeigeprojekt für die Plattform werden. The DAO stellte eine Art Venturecapital-Fonds dar. Das eingesammelte Geld sollte anderen Projekten oder Firmen zur Verfügung gestellt werden, um sich zu finanzieren. Die Investoren von The DAO hätten auf Basis ihrer im Crowdsale erworbenen Tokens abstimmen können, welches Projekt Geld bekommen sollte. Dafür wären sie später an den Gewinnen des Projekts beteiligt gewesen. Dies alles sollte über Smart Contracts ermöglicht werden.

Leider kam das nicht so, denn The DAO hatte in seinem Code eine Schwachstelle, die es Angreifern ermöglichte, Ether der Investoren im Wert von 50 Millionen Dollar unter ihre Kontrolle zu bekommen. Dieser Hack ging mit einem massiven Imageverlust für Ethereum einher und setzte die Entwickler stark unter Druck. Daher entschieden sich die Ethereum-Entwickler mithilfe eines Hard Fork, einen Block einzufügen, der alle

Ether der Investoren inklusive der gestohlenen Ether in einen neu aufgesetzten Smart Contract verschob. Dieser erlaubte den Investoren nun, ihr Geld zurückzufordern. Das stellte einen starken Eingriff in die Ethereum-Blockchain dar, denn die Aktion verstieß eigentlich gegen die grundlegenden Prinzipien der fälschungssicheren Technologie. Proteste der Community ließen nicht lange auf sich warten, und die Ethereum-Miner teilten sich in zwei Lager. Ein Teil unterstützte die abgeänderte Ethereum-Blockchain, die fortan als Ethereum weitergeführt wurde. Der andere Teil entschied sich dazu, die alte Blockchain weiterzuführen. Diese existiert heute immer noch unter dem Namen *Ethereum Classic*.

Im Laufe der Zeit entstanden, etwa mit dem chinesischen Projekt *NEO*, dem deutschen Projekt *Lisk* oder dem Projekt *EOS*, noch weitere Projekte, die eine Blockchain 2.0 ermöglichen.

2.2.5 Gegenwart und Zukunft

Im Jahr 2017 erreichte die Kryptowährung Bitcoin eine große mediale Aufmerksamkeit. In der ersten Jahreshälfte zog der Kurs stark an und knackte im Oktober mit 5.000 Dollar eine erste Höchstmarke. Im Dezember 2017 schnellte der Kurs schließlich auf über 19.000 Dollar hoch. Hierdurch wurden die Politik, die Wirtschaft, aber auch viele Privatanwender auf Bitcoin und die dahinterstehende Technologie aufmerksam. Davon profitierten auch Währungen wie Litcoin, Ethereum und viele neue Projekte, die in Form von ICOs aus dem Boden schossen. Obwohl die Kurse ab dem Jahr 2018 wieder zu fallen begannen, bleibt der Blockchain-Hype ungebrochen. Dies führt zu einer enormen Nachfrage an Blockchain-Entwicklern, die der Arbeitsmarkt aktuell nicht bedienen kann (*https://techcrunch.com/2018/02/14/blockchain-engineers-are-in-demand/*).

Forschung und Wirtschaft beschäftigen sich derzeit mit der Entwicklung der Blockchain 3.0. Auch wenn manche Projekte diesen Titel für sich beanspruchen, herrscht noch keine Einigung darüber, was die Blockchain 3.0 eigentlich ist. Hier und da wird das Projekt IOTA (siehe Kapitel 20) bereits als Blockchain 3.0 bezeichnet, andere Autoren sind dagegen der Meinung, dass es noch keinen würdigen Vertreter für diese Bezeichnung gäbe. So müsse die Weiterentwicklung Probleme der aktuellen Variante, wie die Speicherproblematik oder die Skalierbarkeit, lösen oder auch neue Anwendungsfälle in verschiedenen Branchen mit sich bringen. Die Zukunft wird zeigen, welches Projekt am Ende den Stempel der Blockchain 3.0 aufgedrückt bekommt. Durch die stetig wachsende Vielfalt im Blockchain-Umfeld wird es allerdings vermutlich immer schwieriger werden, die Ausprägungen der Blockchain in solche Evolutionsstufen zu pressen. Beim Wettlauf um die neuesten Applikationen

und Innovationen entstehen auch regelmäßig neue Konsensalgorithmen. Auf die bekanntesten werden wir in Abschnitt 3.4 näher eingehen.

Eine weitere Entwicklung, über die derzeit häufig diskutiert wird, ist das Projekt *Hyperledger*. Das Open-Source-Projekt wurde 2015 von der Linux Foundation ins Leben gerufen und fortan stetig weiterentwickelt. Mit dem immer stärker werdenden Interesse von Unternehmen an der Blockchain-Technologie rückt auch Hyperledger immer mehr in den Fokus. Das Projekt selbst besteht aus mehreren Unterprojekten, in denen Blockchains und Tools entwickelt werden, die auf die besonderen Bedürfnisse von Unternehmen zugeschnitten sind. Viele große Unternehmen bauen mittlerweile auf Unterprojekten von Hyperledger auf und beteiligen sich aktiv an der Weiterentwicklung.

Das bekannteste Unterprojekt nennt sich *Fabric*. Es ist eine flexible Plattform, die dank verschiedener Plug-and-play-Komponenten stark individualisiert werden kann. So können viele verschiedene Organisationen Teil eines Blockchain-Netzwerks werden und gegenseitig ihre Transaktionen validieren. Fabric bildet das Zentrum des Hyperledger-Projekts.

Durch das Unterprojekt *Burrow* wird Hyperledger mit Ethereum verknüpft. So wird es möglich, Smart Contracts, die in der Ethereum-Blockchain bestehen, auch in der eigenen Blockchain zu nutzen. Hierbei kommt die bereits erwähnte Programmiersprache Solidity zum Einsatz. Auch sonst orientiert sich Burrow an den Modulen von Ethereum.

Ein weiteres Unterprojekt ist *Sawtooth*. Die mit ihm realisierten Blockchain-Projekte basieren auf dem neuen Konsensalgorithmus *Proof-of-Elapsed-Time* (*PoET*). Dieses wird in Abschnitt 3.4.8 näher vorgestellt. Das Projekt *Indy* konzentriert sich auf Identitätsmanagement in einem Blockchain-Netzwerk.

Zudem bringt das Projekt Hyperledger einige sehr interessante Tools mit. Das bekannteste Tool, *Composer*, erleichtert es Anwendern von Fabric, Applikationen zu entwickeln. Hierfür bringt es die sogenannte Business-Network-Definition mit sich. In dieser können die Anwender einfach zu speichernde Daten, Teilnehmer und Transaktionen definieren und anschließend als Smart Contract oder Applikation exportieren. Das Tool *Explorer* wird ebenfalls mit Fabric genutzt und bietet eine Weboberfläche, mit der verschiedene Informationen über die Blockchain abgerufen werden können (z. B. Transaktionen, Daten, Blöcke).

Wie dieses Kapitel zeigt, hat Satoshi Nakamoto mit seinen Bitcoins die Entwicklung eines gänzlich neuen Technologiezweigs losgetreten. Dieser entwickelt sich mit einer rasanten Geschwindigkeit weiter und bringt beinahe jeden Tag neue Projekte und Innovationen hervor. Es bleibt spannend, zu beobachten, welche dieser Projekte sich am Ende durchsetzen werden.

Kapitel 4

Eine eigene Blockchain erstellen – Grundfunktionen

Dieses Kapitel widmet sich der Datensicht einer Blockchain. Sie beginnen hier, Ihre erste Blockchain zu entwickeln. Dafür verwenden wir in den Beispielen die Programmiersprache Java und werden nach und nach die zuvor erlernten Grundlagen umsetzen.

Im vorherigen Kapitel haben Sie die grundlegenden Prinzipien einer Blockchain kennengelernt. Dazu haben wir Ihnen die benötigten Datenstrukturen und deren Definition erläutert. Zudem haben wir einen kleinen Ausflug in die mathematische Welt der Kryptografie gemacht. Diese Abschnitte beinhalten die Grundlagen, um die Datenhaltung einer Blockchain zu verstehen. Die Grundlagen zu den Konsensmodellen werden Sie erst bei der Implementierung des Netzwerks benötigen, allerdings sollten Sie bereits wissen, was Proof-of-Work bedeutet.

In den folgenden fünf Kapiteln beginnen Sie nun, Ihre erste Blockchain zu implementieren. Dieses Kapitel behandelt dabei lediglich die Funktionen, die für eine lokale Blockchain ohne dezentrale Verteilung benötigt werden. Das ist hauptsächlich die Datenhaltung, allerdings werden Sie auch schon eine erste Variante von Proof-of-Work implementieren.

Sie werden sich zunächst der Datenstruktur einer Blockchain widmen. Anschließend kümmern Sie sich um die persistente Speicherung der Blockchain. Bevor Sie dann neue Blöcke erzeugen und verketten können, müssen Sie noch den Beginn der Blockchain – den Genesis Block – vorbereiten. Sobald die Blockchain gefüllt werden kann, betrachten Sie die Handhabung von eingehenden Transaktionen und dem Konsensmodell bzw. dem Mining.

Voraussetzung für dieses Kapitel sind Kenntnisse der Programmiersprache Java, da wir direkt mit der Implementierung der Blockchain beginnen werden, ohne zuvor eine Einführung in die verwendete Programmiersprache zu geben. Wir empfehlen Ihnen, eine Entwicklungsumgebung zu verwenden, etwa Intelli J IDEA von Jetbrains (*https://www.jetbrains.com/idea/*). Zudem sollten Sie die zuvor vorgestellten Grundkonzepte kennen.

Alle gezeigten Codebeispiele finden Sie auch unter *https://www.rheinwerk-verlag.de/4677*. Diese können Sie gern als Grundlage verwenden, aber wir empfehlen Ihnen, die einzelnen Schritte der Implementierung zunächst selbst zu versuchen.

Bevor Sie nun mit dem eigentlichen Programmieren der Blockchain beginnen, öffnen Sie die Entwicklungsumgebung Ihrer Wahl und erstellen ein neues Projekt. Für die kryptografischen Berechnungen und Hashing-Verfahren empfehlen wir Ihnen die externe Library von *http://www.bouncycastle.org/java.html*, für die Serialisierung und Deserialisierung von Objekten nach und von JSON die externe Library Genson: *http://genson.io*.

Maven

Für die Verwaltung von Abhängigkeiten dient das Build-Tool *Maven*, aber Sie können die Abhängigkeiten natürlich auch so einbinden, wie Sie es gewohnt sind.

Damit Sie sich besser orientieren können, haben wir Ihnen in Abbildung 4.1 eine Übersicht als UML-Diagramm gebaut. In diesem Kapitel werden Sie die Komponente einer *Full Node* inklusive der Subkomponente *Miner* erstellen. Dafür implementieren Sie Schritt für Schritt die elementaren Klassen einer Blockchain.

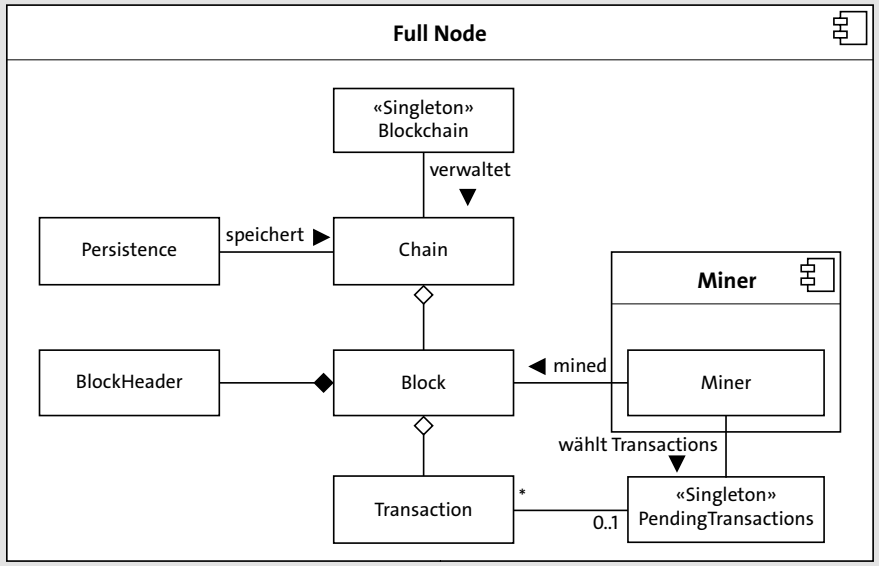

Abbildung 4.1 Eine Übersicht über die zu erstellenden Einheiten und Komponenten dieses Kapitels

Die Implementierung, die Sie in den nächsten fünf Kapiteln umsetzen werden, stellt eine Vereinfachung von Blockchains dar. Der Anspruch dieser Kapitel besteht nicht

darin, eine vollkommen sichere und stabile Blockchain zu implementieren, sondern liegt vielmehr darin, Ihnen einen gut verständlichen Einblick in die Programmierung eigener Blockchains zu geben. Alle implementierten Algorithmen basieren dabei entweder auf den Mechanismen der Blockchain Bitcoin oder der Blockchain Ethereum. Nach diesen fünf Kapiteln sind Sie in der Lage, die grundlegenden Mechanismen der Blockchain-Technologie zu verstehen und zu implementieren. Zudem können Sie ausgehend von dieser Grundlage alle weiteren Verbesserungen selbst durchführen oder recherchieren. Damit Sie nicht während der Implementierung immer wieder in die Grundlagenkapitel zurückblättern müssen, haben wir in diesem Kapitel die verwendeten Begriffe noch einmal passend zur Implementierung kurz erläutert.

4.1 Transaktionen – die kleinste Einheit

Zu Beginn der Blockchain-Technologie drehte sich alles um *Transaktionen*: Wie können Sie sicherstellen, dass diese transparent ablaufen? Wie können Sie verhindern, dass *Double-Spending* betrieben wird? Deshalb beginnt unsere Implementierung auch mit Transaktionen. Dafür erstellen Sie die Klasse `Transaction` in der Datei *models/Transaction.java*.

Da Transaktionen immer etwas von A nach B verschicken, benötigen sie einen Absender und einen Empfänger. Weil beide im Blockchain-Umfeld immer kryptografische Zeichenfolgen sind, könnten Sie hier einfach einen String als Datentyp verwenden. Allerdings werden die kryptografischen Zeichenfolgen üblicherweise immer als byte-Array hinterlegt, damit die Größe der Transaktion minimal bleibt, da Strings oftmals für die Encodierung zusätzliche Bytes benötigen. Für eine bessere Lesbarkeit lässt sich das byte-Array dann ganz einfach in einen String konvertieren. Erstellen Sie dafür beispielsweise eine Hilfsklasse `SHA3Helper` und implementieren Sie die Funktion aus Listing 4.1.

```
public static String digestToHex(byte[] digest) {
    return Hex.toHexString(digest);
}
```

Listing 4.1 Die Methode zum Umwandeln einer kryptografischen Zeichenfolge als byte-Array in einen String mithilfe der Library von Bouncy Castle

Mehrere Empfänger unterstützen

Bei einigen Blockchains ist es möglich, mehr als einen Empfänger innerhalb einer Transaktion anzugeben, beispielsweise bei Bitcoin. Aus Gründen der Übersichtlichkeit beschränken wir uns aber zunächst auf einen Empfänger. Benötigen Sie diese Funktion in Ihrer eigenen Blockchain, können Sie das einfach zusätzlich implementieren, indem Sie anstelle eines einzelnen byte-Arrays eine Liste von byte-Arrays verwenden.

Nachdem nun Absender und Empfänger vorhanden sind, benötigt die Transaktion noch die Menge, die versendet werden soll, sowie das Limit für die Transaktionsgebühr. Zudem kann noch der Basispreis zum Berechnen der Transaktionsgebühr angegeben werden. Die exakte Transaktionsgebühr wird dann erst später vom Miner berechnet, der die Transaktion in seinen Block aufnimmt. Dafür multipliziert der Miner den Basispreis mit den verbrauchten Einheiten. Sollte der Gesamtpreis das angegebene Limit überschreiten, wird die Transaktion abgebrochen. Zudem benötigen Sie noch einen einmaligen ganzzahligen Wert, den sogenannten *Nonce*. Der Nonce wird für jeden Absender bei 0 beginnend hochgezählt und repräsentiert die Anzahl getätigter Transaktionen.

Die zuvor aufgelisteten Attribute werden von einem Nutzer an das Netzwerk geschickt und definieren die Transaktion eindeutig. Allerdings besitzt eine Transaktion noch weitere Attribute, die dann vom Miner erzeugt und hinzugefügt werden. Da aber zu Beginn nur die vorherigen Attribute bekannt sind, benötigen Sie einen passenden Konstruktor wie den in Listing 4.2.

```
public Transaction(byte[] sender, byte[] receiver, double amount,
                   int nonce, double transactionFeeBasePrice,
                   double transactionFeeLimit) {
    this.sender = sender;
    this.receiver = receiver;
    this.amount = amount;
    this.nonce = nonce;
    this.transactionFeeBasePrice = transactionFeeBasePrice;
    this.transactionFeeLimit = transactionFeeLimit;

    this.txId = SHA3Helper.hash256(this);
}
```

Listing 4.2 Der Konstruktor einer Transaktion mit den obligatorischen Attributen. Nach dem Setzen der Attribute wird die ID der Transaktion über kryptografische Verfahren generiert.

Am Ende des Konstruktors in Listing 4.2 wird die ID der Transaktion initialisiert. Analog zu Absender und Empfänger ist die Transaktions-ID auch eine kryptografische Zeichenfolge: das Ergebnis der Hashberechnung mit den sechs obligatorischen Attributen. Für die Hashberechnung nutzen wir die externe Library von Bouncy Castle. Sie können den Code aus Listing 4.3 für alle Hashberechnungen innerhalb der Blockchain verwenden. Da für die Hashberechnung Streams von Java genutzt werden, muss die Klasse `Transaction` das Interface `Serializable` implementieren.

Zusätzlich zu den Attributen, die jede Transaktion eindeutig identifizieren, können weitere Informationen sinnvoll sein. Dazu gehören Zeitstempel, die Größe der Transaktion in Byte sowie die berechnete exakte Transaktionsgebühr. Falls Sie mehr Attri-

bute benötigen, fügen Sie diese einfach der Transaktion hinzu. Denken Sie aber daran, dass die weiteren Attribute mit dem Schlüsselwort `transient` deklariert werden müssen, wenn sie nicht bei der Hashberechnung berücksichtigt werden sollen. Sie sollten alle Attribute bei der Hashberechnung ignorieren, die der Nutzer bei der Erstellung einer Transaktion nicht wissen kann, da sonst auf Basis der Nutzereingaben keine Transaktions-IDs berechnet werden können.


```
ByteArrayOutputStream bos = new ByteArrayOutputStream();
ObjectOutputStream oos = new ObjectOutputStream(bos);
oos.writeObject(o);
oos.flush();
byte[] digest = new SHA3.Digest256().digest(hash256(bos.toByteArray()));
```

Listing 4.3 Die Hashberechnung eines Serializable-Objekts mithilfe der externen Library von Bouncy Castle

Daten in der Blockchain speichern

Bei der Ethereum-Blockchain ist es möglich, nicht nur Tokens von A nach B zu verschicken, sondern auch zusätzlich Daten in der Blockchain zu speichern. Diese Daten werden üblicherweise binärcodiert und an die Transaktion als *Input Data* angehängt. Wenn Sie diese Funktion auch implementieren möchten, benötigen Sie ein weiteres byte-Array. Die Ethereum-Blockchain bezieht diese Input Data dann sogar in die Hashberechnung mit ein.

Zuletzt kann noch ein weiteres optionales Attribut nützlich sein: die *Block-ID* des Blocks, in den die Transaktion eingebettet ist. Da Sie später einen Block-Explorer programmieren, mit dem die Blockchain durchsucht werden kann, ist es sinnvoll, direkt auf den Block einer Transaktion zugreifen zu können. Diese Block-ID muss allerdings zwingend auf `transient` gesetzt werden, da für die Berechnung der Block-ID zunächst alle Hashberechnungen der Transaktionen abgeschlossen sein müssen. Nachdem Sie nun die Transaktionen implementiert haben, kümmern Sie sich als Nächstes um die Struktur der Blöcke.

4.2 Blockheader – der Inhalt der Block-ID

In den Grundlagen in Abschnitt 3.2.2 haben Sie den Inhalt eines Blocks kennengelernt. Der *Blockheader* beinhaltet die Informationen, die für die Hashberechnung der Block-ID genutzt werden. Bevor Sie sich also dem kompletten Block widmen, implementieren Sie zunächst die Klasse `BlockHeader` in der Datei *models/BlockHeader.java*.

Der Blockheader besteht im Gegensatz zur Transaktion nur aus obligatorischen Attributen, die alle in der Hashberechnung berücksichtigt werden. In dieser Blockchain

wird der Blockheader, analog zur Bitcoin-Blockchain, insgesamt 80 Byte groß. Das erste Attribut ist die Versionsnummer. Diese gibt an, unter welcher Version des Sourcecodes der Block generiert wurde. So kann im Fall eines *Forks* zwischen den verschiedenen parallel existierenden Versionen unterschieden werden.

Die Größe des Blockheaders

Der Blockheader wird auf 80 Byte reduziert und besitzt ausschließlich obligatorische Attribute, damit sogenannte *Light Nodes* weniger Speicher zur Verfügung stellen müssen. Diese Light Nodes nutzen nicht die vollständige Blockchain für ihre Berechnungen, sondern speichern lediglich die Blockheader ab. Durch kryptografische Verfahren kann so trotzdem die Gültigkeit von Transaktionen geprüft werden. Damit auch in vielen Jahren jeder Teilnehmer der Blockchain in der Lage ist, diese Validierung durchzuführen, sollte der Blockheader immer so klein wie möglich gehalten werden.

Das zweite Attribut ist der Zeitstempel des Zeitpunkts, an dem der Miner mit der Erstellung des Blocks begonnen hat. Dieser Zeitpunkt muss in der Bitcoin-Blockchain eindeutig nach den Zeitpunkten der vorherigen elf Blöcke liegen. Andere Knoten der Bitcoin-Blockchain lehnen zusätzlich alle Blöcke ab, deren Erstellungszeitpunkt mehr als zwei Stunden in der Zukunft liegt. Die Zeitsynchronisation in einem global verteilten Netzwerk ist nicht trivial, diese Überprüfung ist daher äußerst relevant.

Damit überhaupt eine Verkettung von Blöcken entstehen kann, benötigt jeder Blockheader zudem die ID des vorherigen Blocks. Diese ID ist analog zur Transaktions-ID ein Hashwert. Nutzen Sie hier also wieder ein byte-Array.

Zusätzlich zu den bisherigen Attributen wird noch ein Hash der Liste aller Transaktionen hinterlegt. Dieser Hashwert wird benötigt, damit eine Manipulation einzelner Transaktionen unmöglich wird, da jede Änderung den Hashwert verändern würde. Die Transaktionen sind somit nicht mehr manipulierbar, sobald sie in einen Block eingebettet werden.

Das letzte Attribut stellt den Kern des Minings dar: den sogenannten Nonce. Über diesen kann das Ergebnis der Hashberechnung beeinflusst werden, bis ein gültiger Hash gefunden wurde. Dieser Nonce kann ganz simpel gehalten werden und lediglich eine zufällige Ganzzahl vom Typ `Integer` darstellen. Hierbei geht es nur darum, den Blockheader möglichst schnell zu verändern, sodass ein neuer Hash als Ergebnis ermittelt werden kann. Da die Änderung von Transaktionen nicht einfach zulässig ist, wird dem Header eine Zufallszahl beigelegt, die beliebig verändert werden kann.

Berechnen Sie nun die Gesamtgröße Ihres Blockheaders, ergeben sich die zuvor erwähnten 80 Byte. Die Versionsnummer und der Nonce vom Typ `Integer` haben in Java jeweils eine Größe von 4 Byte. Der Zeitstempel des Typs `Long` ist 8 Byte groß. Der Hashwert des vorherigen Blocks und der Hash der Transaktionsliste haben jeweils eine Länge von 32 Byte:

4 + 4 + 8 + 32 + 32 = 80 Byte

Die Klasse `BlockHeader` besteht somit nur aus Attributen, Konstruktoren sowie Getter- und Setter-Methoden.

Ziel und Difficulty der Blockchain

In den Grundlagen haben Sie bereits erfahren, dass das Ziel, das für die Bestimmung der Difficulty im Netzwerk verwendet wird, auch Bestandteil des Blockheaders ist. Dies ist bei den Produktiv-Blockchains sehr wichtig, da dort die Difficulty nach festgelegten Zeitabständen dynamisch angepasst wird. Ist das Ziel nicht im Blockheader, wären alte Blöcke mit einem anderen Ziel ungültig. Um es hier einfacher zu halten, implementieren Sie mit uns eine Blockchain mit einer gleichbleibenden Difficulty, daher muss das Ziel nicht Teil des Blockheaders sein.

4.3 Die Blöcke verketten

Bevor Sie die eigene Blockchain aufbauen, klären wir zunächst, welche Informationen in den Blöcken benötigt werden. Legen Sie dafür eine Datei *models/Block.java* an und beginnen Sie mit der Klasse `Block`.

In jeden Block wird die *Magic Number* eingetragen, die zu Ihrer Blockchain gehört. Die Magic Number ist in diesem Fall ein Attribut vom Datentyp `Integer`. Zudem besitzt ein Block eine Größe in Byte. Da nur ganze Bytes gefüllt werden müssen, können Sie auch hier einen Integer verwenden.

Den in Abschnitt 4.2 implementierten Blockheader benötigen Sie nun natürlich auch. Ein Block kann nicht ohne Blockheader existieren, Sie müssen also im Konstruktor des Blocks immer einen neuen Blockheader anlegen. Listing 4.4 zeigt den Konstruktor eines Blocks. Da ein Blockheader immer den Zeitstempel des Zeitpunkts beinhaltet, an dem der Miner mit dem Block begonnen hat, kann der Blockheader direkt mit dem Zeitstempel erzeugt werden.

```
public Block(byte[] previousHash){
    this.blockSize = 92; //80 Byte Blockheader + 3*4 Byte für Attribute
    this.transactions = new ArrayList<>();
    this.transactionCount = this.transactions.size();
    this.blockHeader = ↩
        new BlockHeader(System.currentTimeMillis(), previousHash);
}
```

Listing 4.4 Der Konstruktor eines Blocks setzt zu Beginn die Größe des leeren Blocks: die Größe des Blockheaders zuzüglich der Größe der Metadaten.

Wie in Listing 4.4 zu sehen ist, benötigt der Block noch zwei weitere Attribute: die Anzahl der Transaktionen und die Liste der Transaktionen. Da die Größe des Blocks mit der Anzahl Transaktionen steigt, muss beim Hinzufügen einer neuen Transaktion auch immer die Größe des Blocks aktualisiert werden. Jede Transaktion ist in Ihrer Blockchain mit den von uns vorgegebenen Datentypen 128 Byte groß, also addieren Sie der bisherigen Größe immer 128 hinzu. Listing 4.5 zeigt, dass der Blockheader beim Hinzufügen einer Transaktion ebenfalls aktualisiert werden muss, da eine weitere Transaktion den Hash der Transaktionsliste verändert. Zudem muss auch die Transaktionsanzahl aktualisiert werden.

Der Block beinhaltet neben den Getter- und Setter-Methoden auch noch weitere Methoden, um Informationen des Blockheaders durchzureichen. Sie benötigen zwingend eine Methode, die den Hash des Blocks bzw. die Block-ID zurückgibt. Die Block-ID ist allerdings kein eigenes Attribut, sondern ein Synonym für den Hash des Blockheaders.

```
public void addTransaction(Transaction transaction) {
    this.transactions.add(transaction);
    this.transactionCount++;
    this.blockHeader.setTransactionListHash(getTransactionHash());
    this.blockSize += 128;
}
```

Listing 4.5 Die Methode zum Hinzufügen einer Transaktion muss sowohl den Transaktionszähler und die Blockgröße als auch den Blockheader aktualisieren.

Blockhöhe und Coinbase

Bei der Bitcoin-Blockchain gibt es weitere Attribute innerhalb von Blöcken, beispielsweise die Blockhöhe oder die Coinbase. Diese Attribute sind aber für den jetzigen Stand Ihrer Blockchain nicht obligatorisch. In Abschnitt 7.1.3 werden Sie dann die Coinbase für Blöcke ergänzen.

Nachdem Sie nun Blöcke inklusive Blockheader und Transaktionen erzeugen und nutzen können, fehlt Ihnen nur noch die Chain selbst. Hierfür legen Sie die Klasse `Chain` in der Datei *models/Chain.java* an.

Die Klasse `Chain` ist sehr simpel und besitzt recht wenige Attribute. Sie benötigt nur eine Netzwerk-ID und eine Liste von Blöcken. Im Gegensatz zu den sonstigen IDs einer Blockchain ist die Netzwerk-ID lediglich ein Attribut vom Datentyp `Integer`. Diese ID ermöglicht, mehrere Netzwerke der gleichen Blockchain anzubieten – beispielsweise ein Live- und ein Testnetzwerk. Sie können zum jetzigen Zeitpunkt für die Speicherung der Blöcke eine einfache `ArrayList` verwenden. Allerdings benötigen

Sie bald eine Liste, die nebenläufige Zugriffe auf die Chain zulässt, weshalb wir empfehlen, eine `CopyOnWriteArrayList` zu nutzen.

Nun sind Sie in der Lage, Ihre Transaktionen in Blöcke zu packen und diese Blöcke zu einer Kette zu verknüpfen. Allerdings liegen diese Informationen ausschließlich im Arbeitsspeicher Ihres Knotens. Deshalb kümmern Sie sich als Nächstes um die persistente Speicherung Ihrer Blockchain.

Blockchains ohne Blöcke

Nicht jede Blockchain besteht aus Blöcken. So ist beispielsweise der Tangle von IOTA eine alternative Technologie, die ohne Blöcke und lediglich mit Transaktionen funktioniert. In Abschnitt 20.5.3 am Ende des Buchs erklären wir kurz die Funktionsweise eines Tangles.

4.4 Die Blockchain auf die Festplatte speichern

Für die persistente Speicherung gibt es viele Möglichkeiten. Wir entschieden uns hier für eine gut lesbare Variante: Speichern Sie deshalb JSON-Dateien auf die Festplatte. Da für den Versand von Blöcken übers Netzwerk sowieso die Serialisierung von Blöcken und Transaktionen in JSON-Objekte notwendig wird, können Sie diese Repräsentation auch beim Persistieren nutzen.

Für die Serialisierung und Deserialisierung der Blöcke empfehlen wir die externe Library *Genson*. Genson setzt voraus, dass jedes Objekt einen Standardkonstruktor besitzt und alle Attribute, die serialisiert bzw. deserialisiert werden, über eine Getter- und eine Setter-Methode verfügen. Falls das auf Ihre Klassen noch nicht zutrifft, sollten Sie das also bei den Klassen `Chain`, `Block`, `BlockHeader` und `Transaction` nachholen.

Für die Implementierung der Logik erstellen Sie die Klasse `Persistence` in der Datei *persistence/Persistence.java*. Diese benötigt lediglich zwei Attribute: das Character Encoding und den Ordnerpfad, unter dem die Blockchain abgespeichert werden soll. Nutzen Sie UTF-8 als Encoding und legen Sie die Blöcke im Ordner *chains* ab.

Im Wesentlichen benötigen Sie eine Methode zum Laden und eine Methode zum Abspeichern der Blockchain. Listing 4.6 zeigt, wie Sie eine Blockchain abspeichern können. Die Methode durchläuft die Chain und speichert jeden Block einzeln in eine Datei ab. Die Block-ID bildet dabei den Dateinamen. Da die Block-ID intern als byte-Array implementiert ist, sollten Sie sie zuvor mit einer Hilfsfunktion in einen String umwandeln, damit die ID lesbar wird.

```
public void writeChain(Chain chain) throws IOException {
    for (Block block : chain.getBlocks()) {
        String id = SHA3Helper.digestToHex(block.getBlockHash());
```

```
        writeBlock(block, chain.getNetworkId(), id);
    }
}
```

Listing 4.6 Die Methode zum persistenten Abspeichern der Blockchain. Sie sorgt dafür, dass jeder Block in eine eigene Datei gespeichert wird.

In Listing 4.6 wird für jeden Block die Methode `writeBlock` aufgerufen. Diese Methode ist in Listing 4.7 zu sehen und erstellt für jeden Block eine neue Datei. Der Dateipfad wird in der Methode `getPathToBlock` anhand der Netzwerk-ID und der Block-ID bestimmt. Jeder Block wird mithilfe der Library Genson in ein JSON-Objekt umgewandelt und über den `OutputStreamWriter` auf die Festplatte gespeichert.

```
private void writeBlock(Block block, int networkId, String id) {
    File file = new File(getPathToBlock(networkId, id));
    OutputStreamWriter outputStream = ⮌
        new OutputStreamWriter(new FileOutputStream(file), encoding);
    Genson genson = new Genson();
    genson.serialize(block, outputStream);
}
```

Listing 4.7 Die Methode serialisiert jeden Block in ein JSON-Objekt mithilfe der Library Genson. Anschließend wird das JSON-Objekt auf die Festplatte geschrieben.

Das Laden einer gespeicherten Blockchain funktioniert analog zum Abspeichern. Hier wird der Pfad angegeben, unter dem die gespeicherte Blockchain zu finden ist, und anschließend werden alle Dateien bzw. Blöcke, die sich in diesem Ordner befinden, der Reihe nach eingelesen und der Blockchain hinzugefügt. Listing 4.8 zeigt diesen Ladevorgang. Für das Deserialisieren der JSON-Objekte in die konkreten Blöcke ist es wichtig, dass die Klasse `Block` einen Standardkonstruktor besitzt. Zudem muss zwingend eine Setter-Methode für den Zeitstempel im Blockheader vorhanden sein. Fehlt dieser, wird eine Blockchain mit falschem Zeitstempel geladen, die vom Rest des Netzwerks abweichen würde.

```
private Block readBlock(File file) {
    InputStreamReader inputStream = ⮌
        new InputStreamReader(new FileInputStream(file), encoding);
    Genson genson = new Genson();
    return genson.deserialize(inputStream, Block.class);
}
```

Listing 4.8 Die Methode liest einen Block im JSON-Format von der Festplatte und deserialisiert diesen in ein Block-Objekt.

Bitte achten Sie darauf, dass alle Attribute, die in Hashberechnungen berücksichtigt werden, zwingend eine Getter- und eine Setter-Methode haben. Ansonsten würden beim Speichern und Laden der Blockchain wichtige Informationen verloren gehen, und die Knoten im Netzwerk würden die Blöcke aufgrund falscher Hashwerte ablehnen. Da Sie nun eine Blockchain auf der Festplatte abspeichern können, wird es Zeit, eine eigene Blockchain zu starten. Allerdings haben Sie hierbei noch das Problem, dass jeder Block den Hash des vorherigen Blocks benötigt. Sie brauchen also einen besonderen ersten Block für Ihre Blockchain.

Reihenfolge beim Einlesen der Blockchain

Die Block-ID als Dateinamen zu nutzen, ist einfach und performant zu verwenden. Allerdings kann dadurch die Reihenfolge der Blöcke beim Einlesen nicht performant garantiert werden, da die Dateien in alphabetischer Reihenfolge eingelesen werden. Nach dem Einlesen müssten Sie die Blöcke anhand der Vorgänger-ID neu sortieren, um im Arbeitsspeicher wieder die korrekte sequenzielle Reihenfolge herzustellen. Eine Blockhöhe oder Blocknummer als Dateiname kann hier Abhilfe schaffen.

4.5 Der Genesis Block – die Entstehung einer Blockchain

Der erste Block einer Blockchain wird als der *Genesis Block* bezeichnet. Der Genesis Block ist der einzige Block, der keine ID eines Vorgängers benötigt. Er bekommt üblicherweise den Hashcode `0x0` anstelle einer Vorgänger-ID. Der Genesis Block kann genutzt werden, um beispielsweise eine eigene private Blockchain zu erstellen. So kann von jeder existierenden Blockchain eine eigene private Version erzeugt werden, wenn der Genesis Block ausgetauscht wird.

Sie müssen auch so einen Genesis Block zur Verfügung stellen. Listing 4.9 zeigt die minimale Implementierung eines Genesis Blocks. Das Ethereum-Projekt bietet zusätzlich die Möglichkeit, Accounts mit Guthaben zu initialisieren. Sofern Sie das in Ihrer Implementierung umsetzen, können Sie direkt zu Beginn einen Account mit einer Menge an Tokens anlegen, damit Sie nicht selbst minen müssen.

```
public class GenesisBlock extends Block {
    public static byte[] ZERO_HASH = new byte[32];

    public GenesisBlock() {
        super(ZERO_HASH);
    }
}
```

Listing 4.9 Eine minimale Implementierung eines Genesis Blocks

Startguthaben

Da eine Transaktion im Ethereum-Netzwerk immer einen Absender und einen Empfänger benötigt, gibt es dort im Genesis Block extra Funktionalitäten, um einen Account mit Startguthaben zu versorgen, ohne dass es einen passenden Sender gibt.

4.6 Ausstehende Transaktionen

In diesem Abschnitt kümmern Sie sich um die Verwaltung von ausstehenden Transaktionen. Da die Suche nach einem passenden Nonce für den nächsten Block einiges an Zeit beansprucht, können nicht alle Transaktionen im Netzwerk sofort verarbeitet werden. Die Transaktionen, die nicht sofort verarbeitet werden können, werden als ausstehend bezeichnet und in eine Warteschlange eingereiht.

Da jede Transaktion einen Basispreis für die Berechnung der Transaktionsgebühr besitzt, werden üblicherweise die Transaktionen mit dem höchsten Basispreis bevorzugt von Minern behandelt. Der Miner, der erfolgreich einen Block abschließt, erhält die beinhaltete Transaktionsgebühr und möchte natürlich möglichst viel davon. Also wird ein Miner immer versuchen, die Transaktionen mit dem höchsten Basispreis in seinen Block einzubetten.

Zur Verwaltung der ausstehenden Transaktionen legen Sie die Klasse `PendingTransactions` in der Datei *logic/PendingTransactions.java* an. Die Klasse besitzt nur ein Attribut: eine `PriorityQueue` mit dem generischen Typ `Transaction`. Im Konstruktor wird die `PriorityQueue` dann initialisiert, wofür Sie einen `Comparator` benötigen. Der `Comparator` in Listing 4.10 soll die Transaktionen so priorisieren, dass diejenigen mit dem höchsten Basispreis als Erstes genutzt werden.

```
public class TransactionComparatorByFee implements Comparator<Transaction> {
    @Override public int compare(Transaction o1, Transaction o2) {
        int result = 0;
        if (o2.getBasePrice() - o1.getBasePrice() < 0.0) {
            result = -1;
        } else if (o2.getBasePrice() - o1.getBasePrice() > 0.0) {
            result = 1;
        }
        return result;
    }
}
```

Listing 4.10 Der Comparator kann von einer PriorityQueue genutzt werden, um die Transaktionen mit dem höchsten Basispreis für die Transaktionsgebühr zu bevorzugen.

Sobald das Netzwerk einer Blockchain aktiv ist, versuchen viele Miner gleichzeitig, den nächsten Block zu generieren. Dabei kann es passieren, dass ein anderer Miner bereits die Transaktionen aufgebraucht hat, die der eigene Knoten auch in seinen Block packen will. Deshalb benötigt die Klasse `PendingTransactions` die Methode aus Listing 4.11 zum Löschen von Transaktionen. Da jeder Knoten neue Blöcke über das Netzwerk mitgeteilt bekommt, wird das Löschen aus der Warteliste einfach bei jedem neuen Block aufgerufen.

```
public void clearPendingTransactions(Block block) {
    for (Transaction transaction : block.getTransactions()) {
        pendingTransactions.remove(transaction);
    }
}
```

Listing 4.11 Die Methode zum Entfernen von Transaktionen aus der Warteliste. Diese wird beispielsweise benötigt, falls Transaktionen schon in einem anderen Knoten verwendet wurden.

Zudem brauchen Sie noch eine Methode, die eine Liste von Transaktionen für den nächsten Block zur Verfügung stellt. Je nach Blockgröße und je nach Anzahl der verfügbaren Transaktionen kann diese Liste unterschiedlich lang sein.

```
public List<Transaction> getTransactionsForNextBlock() {
    List<Transaction> nextTransactions = new ArrayList<>();
    int transactionCapacity = SizeHelper.calculateTransactionCapacity();

    PriorityQueue<Transaction> tmp = new PriorityQueue<>(transactions);
    while (transactionCapacity > 0 && !tmp.isEmpty()) {
        nextTransactions.add(tmp.poll());
        transactionCapacity--;
    }
    return nextTransactions;
}
```

Listing 4.12 Die Methode wählt die Transaktionen für den nächsten Block.

Listing 4.12 zeigt die Methode, die die nächsten Transaktionen bereitstellt. Wichtig ist, dass die Methode nicht die Transaktionen aus der `PriorityQueue` entfernt, sondern zuvor eine Kopie der `PriorityQueue` anlegt. Ein Iterator wäre nicht möglich, da dieser nicht in der Reihenfolge der Prioritäten über die Queue laufen würde. Aber warum können Sie die Transaktionen nicht einfach entfernen?

Kommt ein neuer Block bei Ihrem Knoten über das Netzwerk an, müssen alle Transaktionen, die bereits im neuen Block enthalten sind, aus der Warteliste entfernt werden. Würden Sie also nicht auf einer temporären Queue arbeiten, müssten Sie

prüfen, welche der Transaktionen bereits im neuen Block enthalten sind und welche nicht. Die Kopie der Queue vereinfacht somit die Implementierung.

Leere Blöcke minen

Bei einigen Blockchains ist es prinzipiell möglich, leere Blöcke zu minen. Die Miner verzichten dabei auf die Transaktionsgebühren der Nutzer und versuchen, einen Block ohne Transaktionen an die Blockchain anzuhängen. Dies hat den Vorteil, dass sich der Miner den Aufwand spart, der mit den ausstehenden Transaktionen entsteht.

Wenn alle Miner nur leere Blöcke erzeugen, würde die jeweilige Blockchain natürlich nicht mehr nutzbar sein, da keine Transaktionen mehr abgearbeitet werden. Allerdings ist gerade am Anfang einer neuen Blockchain wichtig, dass leere Blöcke möglich sind, da die Miner sonst oft im Leerlauf auf neue Transaktionen warten müssten. Dies verbraucht Strom, ohne dass die Miner eine Belohnung bekommen, wodurch sie früher oder später zu einer anderen Blockchain wechseln, um die Zeit sinnvoller zu nutzen.

4.7 Die Difficulty einer Blockchain

Dieser Abschnitt kümmert sich um die Verwaltung der *Difficulty* einer Blockchain sowie um Optimierungen für den performanten Zugriff auf einzelne Blöcke oder Transaktionen innerhalb der Blockchain. Legen Sie zunächst die Klasse `Blockchain` in der Datei *logic/Blockchain.java* an.

Die Klasse `Blockchain` kapselt einerseits den direkten Zugriff auf unsere Klasse `Chain` und stellt andererseits nützliche Funktionen zur Verfügung, die das Durchsuchen der Chain vereinfachen. So kann eine Trennung der Logik von der Datenstruktur ermöglicht werden. Am Ende werden Sie eine Klasse haben, die einen Index für bereits erstellte Blöcke und Transaktionen zur Verfügung stellt und zudem die Difficulty Ihrer Chain regelt.

Sie benötigen ein Attribut für die Difficulty des Datentyps `BigInteger`. Des Weiteren verwaltet die Klasse Ihre `Chain`. Für das performante Durchsuchen bieten sich zwei Maps an, da Sie so in konstanter Zeit über die Block-ID bzw. Transaktions-ID auf die jeweiligen Elemente zugreifen können. Implementieren Sie eine Map für die Blöcke und eine für Transaktionen.

Listing 4.13 zeigt den Konstruktor der Klasse `Blockchain`. Achten Sie unbedingt darauf, die beiden Maps nicht als klassische `HashMap` zu initialisieren, da später sowohl der Miner als auch die Web-API nebenläufig oder parallel auf diese zugreifen müssen. Sie müssen also dafür sorgen, dass der nebenläufige Zugriff sichergestellt ist. Sie können hierfür einfach die `ConcurrentHashMap` von Java verwenden.


```
public Blockchain() {
    this.chain = new Chain(NETWORK_ID);
    this.blockCache = new ConcurrentHashMap<>();
    this.transactionCache = new ConcurrentHashMap<>();
    this.difficulty = new BigInteger("16000");
}
```

Listing 4.13 Der Konstruktor der Klasse Blockchain. Die Difficulty sowie die Maps für Transaktionen und Blöcke werden hier vorbereitet. Zudem wird die Chain initialisiert, die die Datenstruktur der Blockchain bereitstellt.

Für den Zugriff auf die zugrunde liegende `Chain` können Sie einfach Methoden schreiben, die die Aufrufe durchreichen. Sie benötigen unbedingt eine Methode zum Hinzufügen neuer Blöcke. Denken Sie aber daran, dass Sie die neuen Blöcke und auch deren Transaktionen in die jeweilige Map schreiben.

Die korrekte Difficulty ermitteln

In Listing 4.13 wurde die Difficulty der Blockchain zunächst auf `16000` gesetzt. Da die Difficulty darüber entscheidet, wie viele Blöcke pro Sekunde erzeugt werden können, müssen Sie diese später noch für Ihr System anpassen. Die Difficulty ist so definitiv noch zu leicht, und Ihr Miner würde Tausende Blöcke pro Sekunde erschaffen. Wir haben diese Zahl einfach zufällig gewählt, und Sie werden sich dann im nächsten Abschnitt Schritt für Schritt der korrekten Difficulty annähern.

Für die Difficulty einer Blockchain nutzen Sie den Datentyp `BigInteger`. Das ist nicht zwingend notwendig, allerdings vereinfacht dieser Datentyp einige Überprüfungen. In den Grundlagen haben Sie gelernt, dass beim Proof-of-Work-Verfahren so lange gemined wird, bis ein Hashwert gefunden wurde, der eine vorgegebene Difficulty erfüllt. Verwenden Sie für die Difficulty die Klasse `BigInteger`, damit Sie einfach den Hashwert in einen `BigInteger` umwandeln und vergleichen können, ob dieser kleiner ist als die vorgegebene Difficulty. Ein `BigInteger` verwendet in Java intern ein beliebig großes byte-Array, weshalb diese Umwandlung sehr performant ist. Der Hashwert selbst ist in Ihrem Fall ebenfalls ein byte-Array der Länge 32, wodurch sich `BigInteger` anbietet.

```
public boolean fulfillsDifficulty(byte[] digest) {
    BigInteger temp = new BigInteger(digest);
    return temp.compareTo(difficulty) <= 0;
}
```

Listing 4.14 Die Methode fulfillsDifficulty erwartet einen Hashwert und prüft, ob dieser die Difficulty der Blockchain erfüllt.

Listing 4.14 zeigt, wie einfach die Überprüfung der Difficulty mithilfe der Klasse `BigInteger` umgesetzt werden kann. Der in Abschnitt 4.8 folgende Miner Thread wird diese Methode dann nutzen, um zu entscheiden, ob er einen neuen Block erzeugt hat oder weitersuchen muss.

Für eine spätere Anbindung der Web-API sollten Sie noch Methoden implementieren, die einen Zugriff auf einzelne Blöcke und Transaktionen anhand ihrer ID ermöglichen. Hierfür können Sie entweder mithilfe von Schleifen über die gesamte Chain iterieren, bis Sie das gewünschte Element gefunden haben, oder Sie verwenden die vorbereiteten Maps. Da eine Chain sehr lange werden kann, empfehlen wir immer die Nutzung von Maps.

Je nachdem, welchen Komfort Sie einem späteren Nutzer der Blockchain anbieten möchten, kann eine Methode zum Ermitteln des Nachfolgerblocks relevant werden. Implementieren Sie diese ebenfalls.

4.8 Zeit zu schürfen – der Miner Thread

Nachdem Sie nun alle benötigten Funktionalitäten implementiert haben, müssen Sie sich Gedanken um die Komponente Miner und vor allem den *Miner Thread* machen. Dieser soll schließlich permanent laufen und neue Blöcke erzeugen. Dafür wird er sich von der Klasse `PendingTransactions` neue Transaktionen holen und diese in einen Block verpacken. Beginnen Sie also mit der Klasse `Miner` in der Datei *threads/Miner.java*.

Da Ihr Miner später in einem eigenen Thread laufen wird, nutzen Sie das Interface `Runnable` und erstellen die Methode `run`. Der Mining-Algorithmus ist an sich sehr überschaubar. Zunächst muss ein neuer Block erstellt werden, wofür der Hash des vorherigen Blocks sowie ausstehende Transaktionen geladen werden. Anschließend wird die Nonce im Blockheader so lange erhöht, bis der neue Hashwert des Blocks die Difficulty der Blockchain erfüllt. Ist die Difficulty erfüllt, wird der Block der Chain angehängt, und der Prozess beginnt von vorne.

Listing 4.15 zeigt den Algorithmus in gekürzter Form. Exception Handling kann für den Fall, dass die Nonce über den Integer-Wertebereich hinausläuft, noch hinzugefügt werden. Dies ist aber sehr unwahrscheinlich. Zudem sollte noch eine Überprüfung eingebaut werden, die untersucht, ob das Mining neu gestartet werden sollte. Der Neustart ist sinnvoll, wenn neue Transaktionen angekommen sind, die eventuell im Block untergebracht werden können.

```
@Override public void run() {
    while (isMining()) {
        block = getNewBlockForMining();
        while (!cancelBlock && !fulfillsDifficulty(block.getBlockHash())) {
```



```
            block.incrementNonce();
        }
        blockMined(block);
    }
}
```

Listing 4.15 Die Methode run des Miner Threads erhöht die Nonce im Blockheader so lange, bis die Difficulty erfüllt ist. Dann wird der nächste Block begonnen.

Am Ende des Algorithmus wird die Methode `blockMined` aufgerufen. Diese ist einerseits dafür zuständig, den neuen Block der Blockchain hinzuzufügen, andererseits aber auch, um registrierte Listener über einen neuen Block zu informieren.

Diese Listener können später relevant werden, um beispielsweise die durchschnittliche Zeit pro Block zu berechnen. Erstellen Sie hierfür das Interface `MinerListener` in der Datei *threads/MinerListener.java*. Das Interface benötigt vorerst nur eine Methode, wie in Listing 4.16 dargestellt.

```
public interface MinerListener {
    void notifyNewBlock(Block block);
}
```

Listing 4.16 Das Interface MinerListener. Dieses wird genutzt, um die Benachrichtigung über neue Blöcke zu ermöglichen.

Ihre Klasse `Miner` benötigt nun noch eine Liste zum Speichern der Listener sowie eine Methode, über die sich die Listener registrieren können. Die Methode `blockMined` prüft dann zunächst, ob der neue Block Transaktionen beinhaltet. Ist das der Fall ist, wird die Block-ID in den Transaktionen gesetzt, und anschließend wird der Block der Blockchain hinzugefügt. Abschließend werden dann alle Listener über den neuen Block informiert. Dies geschieht wie in Listing 4.17 über eine einfache `for`-Schleife. Wie in Abschnitt 4.6 erwähnt, müssen aber auch noch die im Block enthaltenen Transaktionen aus der Queue der ausstehenden Transaktionen entfernt werden.

Die Block-ID muss auch in den Transaktionen gesetzt werden, da üblicherweise ein Nutzer nur die ID seiner Transaktion kennt. Betrachtet er dann diese Transaktion in einem *Block-Explorer*, hilft die Block-ID dabei, den zugehörigen Block zu finden. Ein Block-Explorer ist ein Tool, mit dem ein Nutzer eine Blockchain entweder komplett durchlaufen oder nach einzelnen Elementen suchen kann.

```
private void blockMined(Block block){
    if (block.getTransactions().size() > 0) {
        for (Transaction transaction : block.getTransactions()) {
            transaction.setBlockId(block.getBlockHash());
        }
```

```
    }
    blockchain.addBlock(block);
    pendingTransactions.clearPendingTransactions(block);

    for (MinerListener listener : listeners) {
        listener.notifyNewBlock(block);
    }
}
```

Listing 4.17 Die Methode prüft zunächst, ob der Block Transaktionen beinhaltet, bei denen dann die Block-ID gesetzt wird. Abschließend werden die Listener informiert.

Nachdem der Miner Thread implementiert wurde, ist die Blockchain zumindest lokal auf Ihrem Rechner lauffähig. Erstellen Sie jetzt die Klasse `MinerTests` in der Datei *test/java/threads/MinerTests.java*. Sie werden nun einen Testfall mithilfe des Frameworks JUnit erstellen, der Ihre Blockchain testet. Sollten Sie das Framework JUnit nicht kennen, können Sie auch eine `main`-Methode erstellen, die das Gleiche macht.

```
@Before
public void setUp() {
    PendingTransactions trx = DependencyManager.getPendingTransactions();
    for(int i = 0; i < 100; i++) {
        String sender = "testSender" + i;
        String receiver = "testReceiver" + i;
        double amount = i * 1.1;
        double transactionFee = 0.0000001 * i;
        trx.addPendingTransaction(new Transaction(
            sender.getBytes(), receiver.getBytes(), amount, 1, ⮐
            transactionFee, 10.0 ⮐
        ));
    }
}
```

Listing 4.18 Die Methode setUp kümmert sich um die Erstellung von Transaktionen, die als ausstehend hinterlegt werden.

Fügen Sie die Abhängigkeit des Frameworks JUnit in Ihre *pom.xml* ein. Anschließend erstellen Sie die Methode `setUp`, die ein paar Transaktionen anlegt und als ausstehend hinterlegt. Diese Transaktionen werden später vom Miner Thread in Blöcke gepackt und zu einer Blockchain zusammengefügt. Machen Sie sich wegen der Informationen der Transaktionen keine großen Gedanken und nutzen Sie wie in Listing 4.18 irgendwelche Daten. Die erste Zeile der Methode ruft den `DependencyManager` auf. Dies ist eine Klasse, die Singleton-Instanzen der Klassen `Blockchain` und `PendingTransactions` verwaltet. Sie können auch diese Klassen direkt als Singleton implementieren.

Als Nächstes fügen Sie der Testklasse das Interface `MinerListener` hinzu, da die Klasse selbst vom Miner Thread benachrichtigt werden soll, wenn ein neuer Block erzeugt wurde. Das Interface zwingt Sie dann dazu, die Methode `notifyNewBlock` zu implementieren. Implementieren Sie diese nach Ihren Vorlieben. Geben Sie beispielsweise die Block-ID auf der Konsole aus.

Jetzt benötigen Sie nur noch einen Testfall, wie in Listing 4.19, den Sie starten können. Zunächst muss der Miner initialisiert und die Testklasse als Listener registriert werden. Anschließend können Sie den Miner Thread starten und den Thread der Tests so lange schlafen legen, bis keine Transaktionen mehr ausstehend sind. Am Ende stoppen Sie den Miner Thread, und der Testfall wird beendet. Während des Mining-Prozesses sollten permanent Block-IDs ausgegeben werden, da der Miner bei jedem neuen Block Ihren Listener benachrichtigt.

```
@Test
public void testMiner() {
    Miner miner = new Miner();
    miner.registerListener(this);
    Thread thread = new Thread(miner);
    thread.start();

    while (DependencyManager.getPendingTransactions() ⮐
                            .pendingTransactionsAvailable()) {
        Thread.sleep(1000);
    }

    miner.stopMining();
}
```

Listing 4.19 Der Testfall, der den Miner startet und die ausstehenden Transaktionen zu einer Blockchain zusammensetzt.

Der Testfall zeigt, dass Ihre Blockchain zumindest lokal auf Ihrem Rechner lauffähig ist. Anhand der Ausgaben auf der Konsole können Sie auch sehen, wie viele Blöcke der Miner Thread pro Sekunde in etwa erzeugen kann. Diese Ausgabe können Sie nutzen, um die Difficulty aus Abschnitt 4.7 an Ihr System anzupassen. Erhöhen Sie die Difficulty so lange, bis ein Block etwa zwei Sekunden braucht, um gemined zu werden. Der Wert der Difficulty wird am Ende sehr, sehr klein sein. Da jeder Hashwert aus 32 Byte besteht, benötigen Sie auch eine 32 Byte große negative Zahl. In unserem Testsystem war die benötigte Difficulty
-5789**555**00000000000000000000000000000000000000000000000000000000000000000000000.
Nutzen Sie diese Difficulty als Anfangswert und variieren Sie zunächst die drei Fünfen, bis Sie eine passende Difficulty für Ihr System gefunden haben.

4.9 Zusammenfassung und Ausblick

In diesem Kapitel haben Sie damit begonnen, Ihre erste Blockchain zu entwickeln. Dafür haben wir zunächst die Datensicht vorgestellt, und Sie haben dann Stück für Stück alle Elemente implementiert, bis Sie am Ende eine lokal lauffähige Blockchain hatten. Eine kurze Zusammenfassung dessen, was Sie aus diesem Kapitel mitnehmen sollten, beinhaltet die folgende Liste:

- Hashwerte können zwar auch als String repräsentiert werden, sind aber klassischerweise byte-Arrays. Für den Nutzer der Blockchain können diese später aber in Strings umgewandelt werden, um bessere Lesbarkeit zu erzeugen.
- Sowohl Transaktionen als auch Blöcke können eine Nonce besitzen. Bei Transaktionen repräsentieren diese später einfach die Anzahl der bisherigen Transaktionen, die der Nutzer mit seinem Account getätigt hat, wohingegen bei Blöcken die Nonce als Proof für die benötigte Rechenzeit verwendet wird.
- Nicht alle Attribute eines Blocks oder einer Transaktion werden für die Hashberechnung verwendet. Bei jedem Attribut muss überlegt werden, ob dieses zur eindeutigen Identifizierung benötigt wird und deshalb berücksichtigt werden soll.
- Die magische Nummer wird in vielen Protokollen und Dateiformaten genutzt, um eine schnelle Identifikation des Typs zu ermöglichen. Auch bei Blockchains nutzen die Knoten diese Nummer, um schnell zu ermitteln, ob die übermittelten Daten zur eigenen Blockchain gehören.
- Die Verkettung der Blöcke zu einer Blockchain entsteht dadurch, dass jeder Block den Hashwert des vorherigen Blocks in sich trägt. Ein Block hat jedoch keinerlei Informationen über den Nachfolger.
- Beim Abspeichern und auch beim Übermitteln von Blöcken sollte immer auf das UTF-8-Character-Encoding geachtet werden.
- Der Genesis Block ist ein besonderer Block, der den Anfang einer Blockchain markiert. Dieser kann genutzt werden, um Accounts oder Wallets von Anfang an mit Tokens zu befüllen.
- Ausstehende Transaktionen werden häufig nach der Höhe des Basispreises der Transaktionsgebühr priorisiert. Es gibt jedoch auch Idealisten, die eine bestimmte Blockchain unterstützen möchten und absichtlich die Transaktionen mit niedrigerem Basispreis bevorzugen.
- Leere Blöcke sind standardmäßig erlaubt bei Blockchains, da es immer kompliziert ist, zu entscheiden, wie lange auf weitere Transaktionen gewartet werden soll, und die Miner versuchen auch mit leeren Blöcken, ihre Belohnungen zu erhalten.

- Die Difficulty wird üblicherweise als `BigInteger` angegeben, da diese immer aus der gleichen Anzahl Bytes bestehen sollte wie ein Hashwert. Das variiert je nach verwendetem Hashing-Verfahren – in unserem Fall sind es 32 Byte.
- Aufgrund nebenläufiger oder paralleler Zugriffe auf die Blockchain sollten alle Maps und Listen so initialisiert werden, dass sie nebenläufig genutzt werden.
- Der Algorithmus des Miners ist nicht sehr komplex. Ein neuer Block wird erstellt, dann mit ausstehenden Transaktionen und dem Hashwert des vorherigen Blocks gefüllt, und anschließend wird der Hash berechnet. Erfüllt der Hash die Difficulty nicht, wird der Nonce im Blockheader erhöht. Das Ganze läuft so lange, bis die Difficulty erfüllt ist und der Block angehängt werden darf.
- Alle Listener werden in der Implementierung über einen neuen Block informiert. Bei einer großen Anzahl Listener könnte das zu Einbußen bei der Performance führen.

Da Sie nun eine lokal lauffähige Blockchain-Instanz implementiert haben, benötigen Sie als Nächstes eine Schnittstelle. Diese Schnittstelle wird einem Nutzer ermöglichen, Transaktionen an die Blockchain zu schicken. Dafür implementieren Sie im nächsten Kapitel eine Web-API, die die Kommunikation mit der Blockchain ermöglicht. Zudem erstellen Sie ein kleines Webinterface zum Versenden von Transaktionen. Anschließend implementieren Sie einen Block-Explorer, damit Sie Ihre eigene Blockchain erkunden können.