1 Spring in the real world

This chapter covers

- What a framework is
- When to use and when to avoid using frameworks
- What the Spring framework is
- Using Spring in real-world scenarios

The Spring framework (shortly, Spring) is an application framework that is part of the Java ecosystem. An *application framework* is a set of common software functionalities that provides a foundation structure for developing an application. An application framework eases the effort of writing an application by taking out the effort of writing all the program code from scratch.

We use Spring in the development of many kinds of applications nowadays, from large backend solutions to automation testing apps. According to many survey reports on Java technologies (like this one of JRebel from 2020: **http:// mng.bz/N4V7**; or this one from JAXEnter: **http://mng.bz/DK9a**), Spring is the most used Java framework today.

Spring is popular, and developers have started to use it more often with other JVM languages than Java as well. In the last few years, we observed an impressive growth of developers using Spring with Kotlin (another appreciated language from the JVM family). In this book, we'll focus on the foundations of Spring, and I'll teach you essential skills for using Spring in real-world examples. To make the subject more comfortable for you and allow you to focus on Spring, we'll use only Java examples. Throughout the book, we'll discuss and apply, with examples, essential skills like connecting to a database, establishing communication between applications, and securing and testing an app.

Before diving into more technical details in the next chapters, let's talk about the Spring framework and where you'll actually use it. Why is Spring so appreciated, and when should you even use it?

In this chapter, we'll focus on what a framework is, referring in particular to the Spring framework. In section 1.1, we discuss the advantages of using a framework. In section 1.2, we discuss the Spring ecosystem with the components you need to learn to get started with Spring. Then I'll take you through possible usages of the Spring framework—in particular, real-world scenarios in section 1.3. In section 1.4, we'll discuss when using frameworks might not be the right approach. You need to understand all these things about the Spring framework before trying to use it. Otherwise, you might try to use a hammer to dig your garden.

Depending on your level, you might feel this chapter difficult. I might introduce some notions that you haven't heard about, and this aspect might disturb you. But don't worry; even if you don't understand some of the things now, they will be clarified later in the book. Sometimes, throughout the book, I'll refer to something said in earlier chapters. I use this approach because learning a framework like Spring doesn't always offer us a linear learning path, and sometimes you need to wait until you get more pieces of the puzzle before you see the complete picture. But in the end, you'll get a clear image, and you'll get the valuable skills you need to develop apps like a professional.

1.1 Why should we use frameworks?

In this section, we discuss frameworks. What are they? How did this concept appear, and why? To be motivated to use something, you need to know how that something brings you value. And that's also the case with Spring. I'll teach you these essential details by sharing the knowledge I gathered from my own experience and by studying and using various frameworks in real-world scenarios, including Spring.

An application framework is a set of functionalities on top of which we build applications. The application framework provides us a broad set of tools and functionalities that you can use to build apps. You don't need to use all the features the framework offers. Depending on the requirements of the app you make, you'll choose the right parts of the framework to use.

Here's an analogy I like for application frameworks. Did you ever buy a piece of furniture from a DIY store like Ikea? Say you buy a wardrobe—you won't get an assembled wardrobe, but the right components you

need to build it and a manual on how to assemble your piece of furniture. Now imagine you ordered a wardrobe, but instead of getting only the right components you need, you get all the possible components you can use to assemble any piece of furniture: a table, a wardrobe, and so on. If you want a wardrobe, you have to find the right parts and assemble them. It's like an application framework. The application framework offers you various pieces of software you need to build your app. You need to know what features to choose and how to assemble them to achieve the right result (figure 1.1).

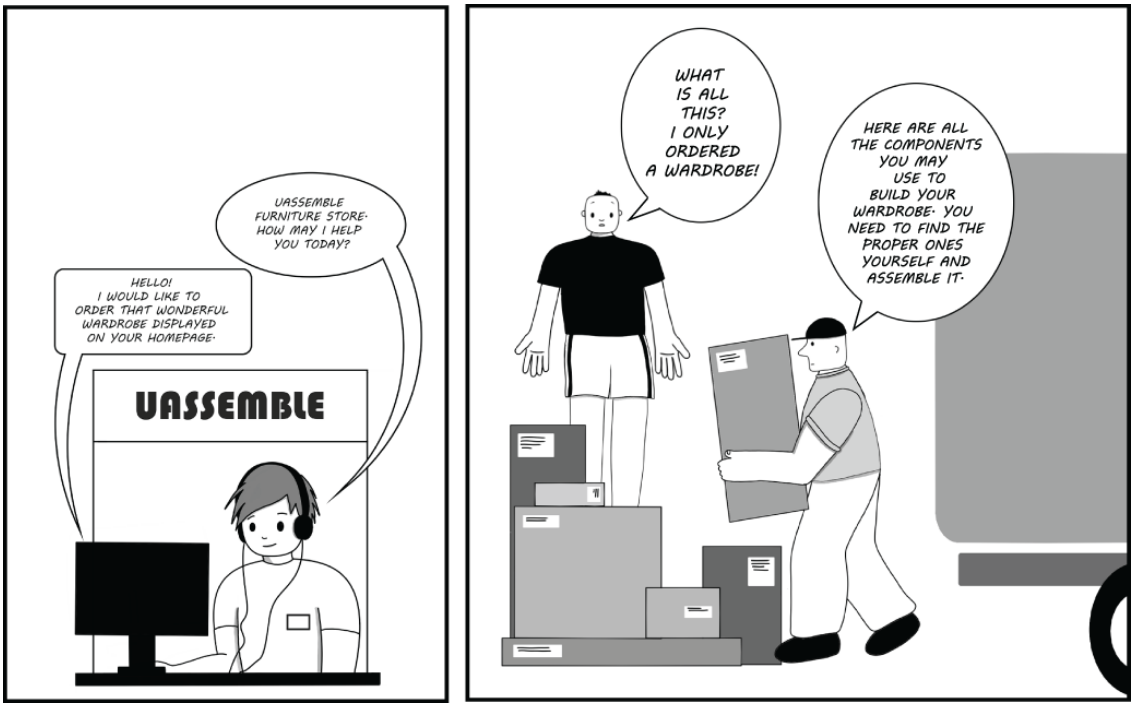

Figure 1.1 David ordered a wardrobe from the UAssemble store. But the store (framework) doesn't deliver to David (the programmer) just the components (software capabilities) he needs to build his new wardrobe (the app). The store ships him all the possible parts he might need to build the wardrobe. It's David's (the programmer's) choice on which components (software capabilities) are right and how to assemble them to get the right result (the application).

The idea of a framework isn't new. Throughout the history of software development, programmers observed they could reuse parts of code they'd written in multiple applications. Initially, when not so many applications were implemented, each application was unique and developed from scratch using a specific programming language. When the software development domain extended, and more and more applications started to be published on the market, it became easier to observe that many of these apps had similar requirements. Let's name a few of them:

- Logging error, warning, and info messages happen in every app.
- Most applications use transactions to process data changes. Transactions represent an important mechanism that takes care of data consistency. We'll discuss this subject in detail in chapter 13.
- Most applications use protection mechanisms against the same common vulnerabilities.
- Most applications use similar ways to communicate with each other.
- Most applications use similar mechanisms to improve their performance, like caching or data compression.

And the list continues. It turns out that the business logic code implemented in an app is significantly smaller than the wheels and belts that make the engine of the application (also often referred to as "the plumbing").

When I say "business logic code," I refer to the code that implements the business requirements of the application. This code is what implements the user's expectations in an application. For example, "clicking on a specific link will generate an invoice" is something users expect to happen. Some code of the application you develop implements this functionality, and this part of code is what developers call the business logic code. However, any app takes care of several more aspects: security, logging, data consistency, and so on (figure 1.2).

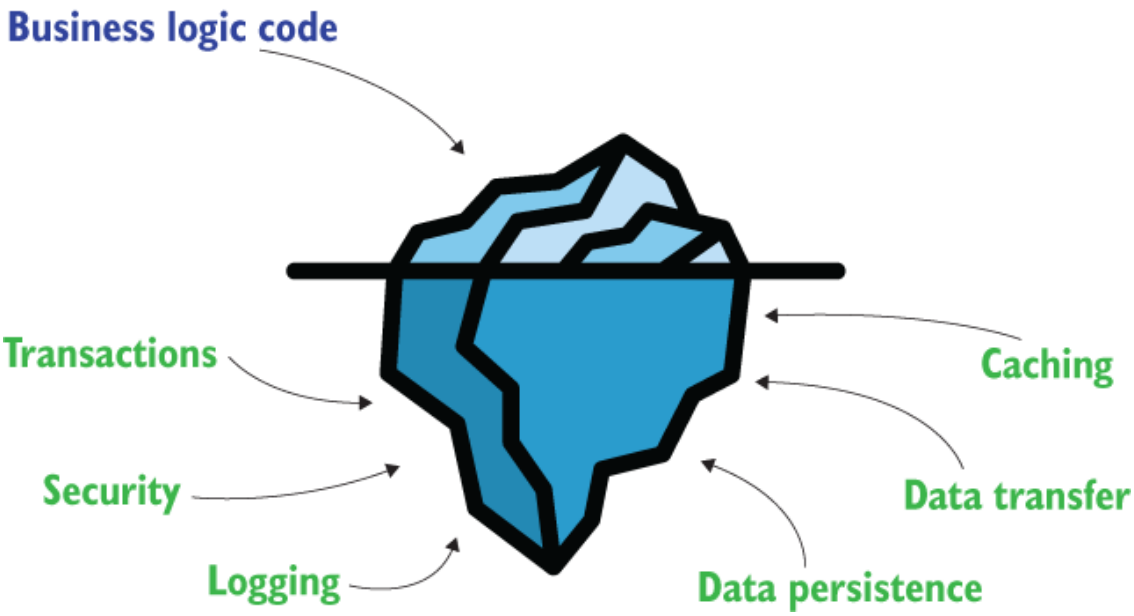

Figure 1.2 The user's perspective is similar to viewing an iceberg. Users mainly observe the results of the business logic code, but this is only a small part of what builds the app's complete functionality. Like an iceberg

that is mostly underwater and hidden from view, we don't see most of the code in an enterprise app because it's provided by dependencies.

Moreover, the business logic code is what makes an application different from another from the functionality point of view. If you take two different apps, say a ridesharing system and a social networking app, they have different use cases.

NOTE A *use case* represents the reason a person uses the app. For example, in a ridesharing app, a use case is “requesting a car.” For an app managing food delivery, a use case is “ordering a pizza.”

You take different actions, but they both need data storing, data transfer, logging, security configurations, probably caching, and so on. Various applications can reuse these nonbusiness implementations. Is it then efficient to rewrite the same functionalities every time? Of course not:

- You spare a lot of time and money by reusing something rather than developing it yourself.
- An existing implementation that many apps already use has fewer chances to introduce bugs, as others have tested it.
- You benefit from the advice of a community because you now have a lot of developers understanding the same functionality. If you had implemented your own code, only a few people would know it.

A story of transition

One of the first applications I worked on was a huge system developed in Java. This system was composed of multiple applications designed around an old-fashioned architecture server, all of them written from scratch using Java SE. The development of this application started with the language about 25 years ago. This was the main reason for its shape. And almost no one could have imagined how big it would become. At that time, more advanced concepts of system architectures didn’t exist, and things in general worked differently from the individual systems due to the slow internet connection.

But time passed, and years later, the app was more like a big ball of mud. For valid reasons I won't cover here, the team decided they had to go to a modern architecture. This change implied first cleaning up the code, and one of the main steps was using a framework. We decided to go with Spring. At that time, we had as an alternative Java EE (now named Jakarta EE), but most members of the team considered it's better to go with Spring, which offered a lighter alternative that was easier to implement and that we also considered easier to maintain.

The transition wasn't an easy one. Together with a few colleagues, experts in their domain and knowledgeable about the application itself, we invested a lot of effort into this transformation.

The result was amazing! We removed over 40% of the lines of code. This transition was the first moment I understood how significant the impact of using a framework could be.

NOTE Choosing and using a framework is linked to the design and architecture of an application. You'll find it useful to learn more about these subjects along with learning the Spring framework. In appendix A, you'll find a discussion about software architectures with excellent resources if you'd like to go into details.

1.2 The Spring ecosystem

In this section, we will discuss Spring and related projects like Spring Boot or Spring Data. You'll learn all about these in this book, and the links among them. In real-world scenarios, it's common to use different frameworks together, where each framework is designed to help you implement a specific part of the app faster.

We refer to Spring as a framework, but it is much more complex. Spring is an ecosystem of frameworks. Usually, when developers refer to the Spring framework, they refer to a part of the software capabilities that include the following:

1. *Spring Core*—One of the fundamental parts of Spring that includes foundational capabilities. One of these features is the Spring context. As you'll learn in detail in chapter 2, the Spring context is a fundamental capability of the Spring framework that enables Spring to manage instances of your app. Also, as part of Spring Core, you find the Spring aspects functionality. Aspects help Spring intercept and manipulate methods you define in your app. We discuss more details of the aspects in chapter 6. The Spring Expression Language (SpEL) is another capability you'll find as part of Spring Core, which enables you to describe configurations for Spring using a specific language. All of these are new notions, and I don't expect you to know them yet. But soon you'll understand that Spring Core holds the mechanisms Spring uses to integrate into your app.
2. *Spring model-view-controller (MVC)*—The part of the Spring framework that enables you to develop web applications that serve HTTP requests. We'll use Spring MVC starting in chapter 7.
3. *Spring Data Access*—Also one of the fundamental parts of Spring. It provides basic tools you can use to connect to SQL databases to implement the persistence layer of your app. We'll use Spring Data Access starting in chapter 13.
4. *Spring testing*—The part holding the tools you need to write tests for your Spring application. We'll discuss this subject in chapter 15.

You can initially imagine the Spring framework as a solar system, where Spring Core represents the star in the middle, which holds all the framework together (figure 1.3).

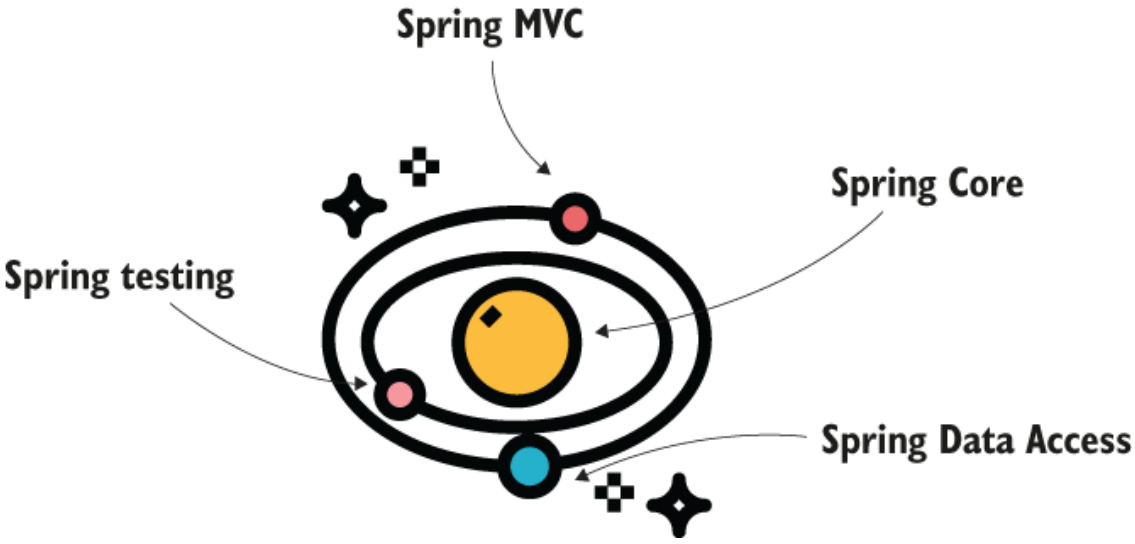

Figure 1.3 You can imagine the Spring framework as a solar system with the Spring Core in the center. The software capabilities are planets

around Spring Core kept close to it by its gravitational field.

1.2.1 Discovering Spring Core: The foundation of Spring

Spring Core is the part of the Spring framework that provides the foundational mechanisms to integrate into apps. Spring works based on the principle *inversion of control (IoC)*. When using this principle, instead of allowing the app to control the execution, we give control to some other piece of software—in our case, the Spring framework. Through configurations, we instruct the framework on how to manage the code we write, which defines the logic of the app. Here's where the "inversion" in IoC comes from: you don't let the app control the execution by its own code and use dependencies. Instead, we allow the framework (the dependency) to control the app and its code (figure 1.4).

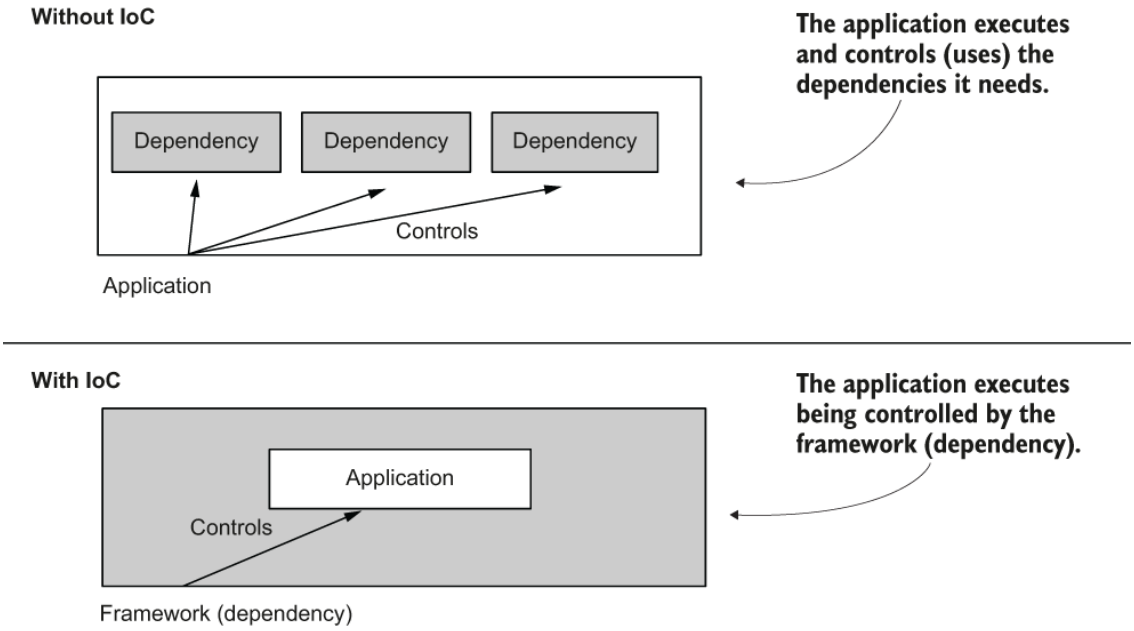

Figure 1.4 Inversion of control. Instead of executing its own code, which makes use of several other dependencies, in case of an IoC scenario, the app execution is controlled by the dependency. The Spring framework controls an app during its execution. Therefore, it implements an IoC scenario of execution.

NOTE In this context the term "controls" refers to actions like "creating an instance" or "calling a method." A framework can create objects of the classes you define in your app. Based on the configurations that you write, Spring intercepts the method to augment it with various features. For example, Spring can intercept a specific method to log any error that might appear during the method's execution.

You will start learning Spring with Spring Core by discussing the Spring IoC functionality in chapters 2 through 5. The IoC container glues Spring components and components of your application to the framework together. Using the IoC container, to which you often refer as the Spring context, you make certain objects known to Spring, which enables the framework to use them in the way you configured.

In chapter 6, we'll continue our discussion with Spring aspect-oriented programming (AOP). Spring can control instances added to its IoC container, and one of the things it can do is intercept methods that represent the behavior of these instances. This capability is called *aspecting* the method. Spring AOP is one of the most common ways the framework interacts with what your app does. This trait makes Spring AOP part of the essentials as well. Part of the Spring Core, we also find resource management, internationalization (i18n), type conversion, and SpEL. We'll encounter aspects of these features in examples throughout the book.

1.2.2 Using Spring Data Access feature to implement the app's persistence

For most applications, it's critical to persist part of the data they process. Working with databases is a fundamental subject, and in Spring, it's the Data Access module that you'll use to take care of data persistence in many cases. The Spring Data Access includes using JDBC, integrating with *object-relational mapping* (ORM) frameworks like Hibernate (don't worry if you don't yet know what an ORM framework is or haven't heard about Hibernate; we'll discuss these aspects later in the book), and managing transactions. In chapters 12 through 14, we'll cover everything needed to get you started with Spring Data Access.

1.2.3 The Spring MVC capabilities for developing web apps

The most common applications developed with Spring are web apps, and within the Spring ecosystem, you'll find a large set of tools that enables you to write web applications and web services in different fashions. You can use the Spring MVC to develop apps using a standard servlet fashion, which is common in a vast number of applications today. In chapter 7, we'll go into more detail on using the Spring MVC.

1.2.4 The Spring testing feature

The Spring testing module offers us a large set of tools that we'll use to write unit and integration tests. There have been many pages written about the testing topic, but we'll discuss everything that is essential to get you started with Spring testing in chapter 15. I'll also refer to some valuable resources you need to read to get all the details of this topic. My rule of thumb is that you're not a mature developer if you don't understand testing, so this topic is one you should care about.

1.2.5 Projects from the Spring ecosystem

The Spring ecosystem is so much more than just the capabilities discussed earlier in this section. It includes a big collection of other frameworks that integrate well and form a larger universe. Here we have projects like Spring Data, Spring Security, Spring Cloud, Spring Batch, Spring Boot, and so on. When you develop an app, you can use more of these projects together. For example, you can build an app using all of Spring Boot, Spring Security, and Spring Data. In the next few chapters, we'll work on smaller projects that make use of various projects of the Spring ecosystem. When I say project, I refer to a part of the Spring ecosystem that is independently developed. Each of these projects has a separate team that works on extending its capabilities. Also, each project is separately described and has its own reference on the Spring official website: **https://spring.io/projects**.

Out of this vast universe created by Spring, we'll also refer to Spring Data and Spring Boot. These projects are often encountered in apps, so it's important to get to know them from the beginning.

EXTENDING THE PERSISTENCE CAPABILITIES WITH SPRING DATA

The Spring Data project implements a part of the Spring ecosystem that enables you to easily connect to databases and use the persistence layer with a minimum number of lines of code written. The project refers to both SQL and NoSQL technologies and creates a high-level layer, which simplifies the way you work with data persistence.

NOTE We have Spring Data Access, which is a module of Spring Core, and we also have an independent project in the Spring ecosystem named

Spring Data. Spring Data Access contains fundamental data access implementations like the transaction mechanism and JDBC tools. Spring Data enhances access to databases and offers a broader set of tools, which makes development more accessible and enables your app to connect to different kinds of data sources. We'll discuss this subject in chapter 14.

SPRING BOOT

Spring Boot is a project part of the Spring ecosystem that introduces the concept of "convention over configuration." The main idea of this concept is that instead of setting up all the configurations of a framework yourself, Spring Boot offers you a default configuration that you can customize as needed. The result, in general, is that you write less code because you follow known conventions and your app differs from others in few or small ways. So instead of writing all the configurations for each and every app, it's more efficient to start with a default configuration and only change what's different from the convention. We'll discuss more about Spring Boot starting in chapter 7.

The Spring ecosystem is vast and contains many projects. Some of them you encounter more often than others, and some you may not use at all if you're building an application without a particular need. In this book, we refer only to the projects that are essential for you to get started: Spring Core, Spring Data, and Spring Boot. You can find a full list of projects that are part of the Spring ecosystem on the official Spring website: **https://spring.io/projects/.**

Alternatives for using Spring

We can't really discuss alternatives to Spring because someone could misunderstand them as alternatives to the entire ecosystem. But for many of the individual components and projects that create the Spring ecosystem, you can find other options like other open source or commercial frameworks or libraries.

For example, let's take the Spring IoC container. Years ago, the Java EE specification was a solution very much appreciated by the developers. With a slightly different philosophy, Java EE (which in 2017 was open

sourced and remade in Jakarta EE, **https://jakarta.ee/**) offered specifications like Context and Dependency Injection (CDI) or Enterprise Java Beans (EJB). You could use CDI or EJB to manage a context of object instances and implement aspects (named "interceptors" in the EE terminology). Also, throughout history, Google Guice (**https://github.com/google/guice**) was an appreciated framework for the management of object instances in a container.

For some of the projects taken individually, you could find one or more alternatives. For example, you could choose to use Apache Shiro (**https://shiro.apache.org/**) instead of Spring Security. Or you could decide to implement your web app using the Play framework (**https://www.playframework.com/**) instead of Spring MVC and Spring-related technologies.

A more recent project that looks promising is Red Hat Quarkus. Quarkus is designed for cloud native implementations and becomes more and more mature with rapid steps. I wouldn't be surprised to see it as one of the lead projects in developing enterprise apps in the Java ecosystem in the future (**https://quarkus.io/**).

My advice for you is to always take into consideration your alternatives. In software development, you need to be open-minded and never trust one solution as being "the one." You'll always find scenarios in which a specific technology works better than another.

1.3 Spring in real-world scenarios

Now that you have an overview of Spring, you're aware of when and why you should use a framework. In this section, I'll give you some application scenarios in which using the Spring framework might be an excellent fit. Too often, I've seen developers only refer to backend applications for using a framework like Spring. I've even seen a trend of restricting, even more, the scenario to backend web applications. While it's true that in plenty of cases we see Spring used in this way, it's important to remember

that the framework isn't limited to this scenario. I've seen teams successfully using Spring in different kinds of applications, such as the development of an automation testing app or even in standalone desktop scenarios.

I'll further describe to you some common real-world scenarios in which I've seen Spring used successfully. These are not the only possible scenarios, and Spring might not work all the time in these cases. Remember what we discussed in section 1.2: a framework is not always a good choice. But these are common cases in which generally Spring is a good fit:

1. The development of a backend app
2. The development of an automation testing framework
3. The development of a desktop app
4. The development of a mobile app

1.3.1 Using Spring in the development of a backend app

A backend application is the part of a system that executes on the server side and has the responsibility of managing data and serving client applications' requests. The users access functionalities by using the client apps directly. Further, the client apps make requests to the backend app to work with the users' data. The backend app might use databases to store data or communicate with other backend apps in different fashions.

You can imagine, in a real-world scenario, that the app would be the backend application managing the transactions in your bank accounts. Users may access their accounts and manage them via a web application (online banking) or a mobile app. Both the mobile apps and the web apps represent clients for the backend application. To manage users' transactions, the backend application needs to communicate with other backend solutions, and part of the data it manages needs to be persisted in a database. In figure 1.5, you can visualize the architecture of such a system.

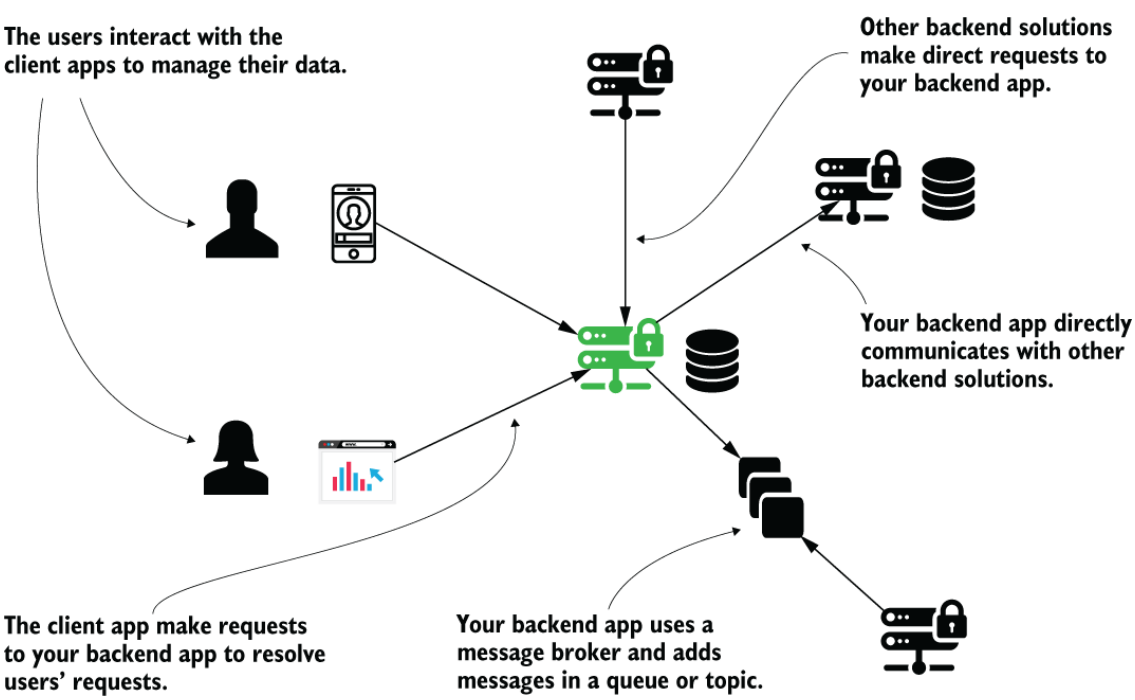

Figure 1.5 A backend app interacts in several ways with other apps and uses databases to manage data. Usually, a backend app is complex and may require the use of various technologies. Frameworks simplify the implementation by providing tools you can use to implement the backend solution faster.

NOTE Don't worry if you don't understand all the details of figure 1.5. I don't expect you to know what a message broker is and not even how to establish the data exchange among the components. What I want you to see is that such a system can become complex in the real world and then understand that projects from the Spring ecosystem were built to help you eliminate this complexity as much as possible.

Spring offers an excellent set of tools for implementing backend applications. It makes your life easier with the different functionalities you generally implement in a backend solution, from integration with other apps to persistence in various database technologies. It's no wonder developers often use Spring for such applications. The framework basically offers you everything you need in such implementations and is an excellent fit for any kind of architectural style. Figure 1.6 indicates the possibilities of using Spring for a backend app.

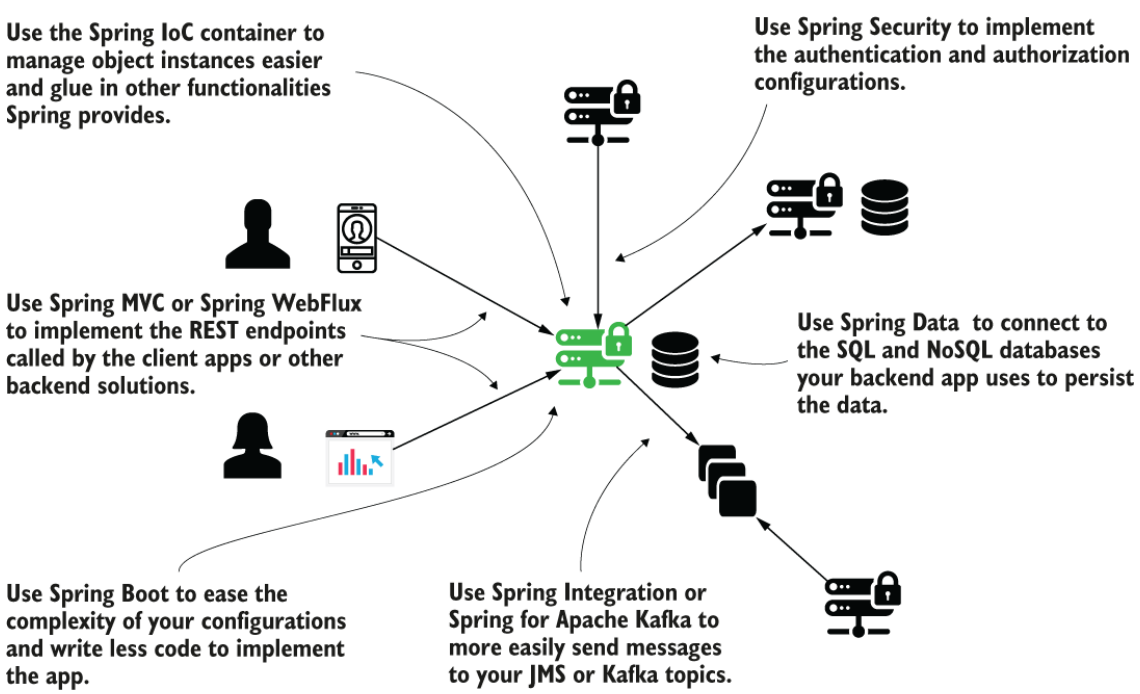

Figure 1.6 The possibilities of using Spring in a backend application are endless, from exposing functionalities that other applications can call to managing the database access, and from securing the application to managing integration though third-party message brokers.

1.3.2 Using Spring in a automation test app

Nowadays, we often use automation testing for end-to-end testing of systems we implement. Automation testing refers to implementing software that development teams use to make sure an application behaves as expected. A development team can schedule the automation testing implementation to frequently test the app and notify the developers if something is wrong. Having such functionality gives developers confidence because they know they'll be notified if they break anything in the existing capabilities of the app while developing new features.

While with small systems you can do the testing manually, it's always a good idea to automate the test cases. For more complex systems, manually testing all the flows isn't even an option. Because the flows are so numerous, it'd require a massive number of hours and too much energy to cover it completely.

It turns out that the most efficient solution is to have a separate team implement an app that has the responsibility of validating all the flows of the tested system. While developers add new functionalities to the system, this testing app is also enhanced to cover what's new, and the teams use it to validate that everything still works as desired. The developers

eventually use an integration tool and schedule the app to run regularly to get feedback as soon as possible for their changes (figure 1.7).

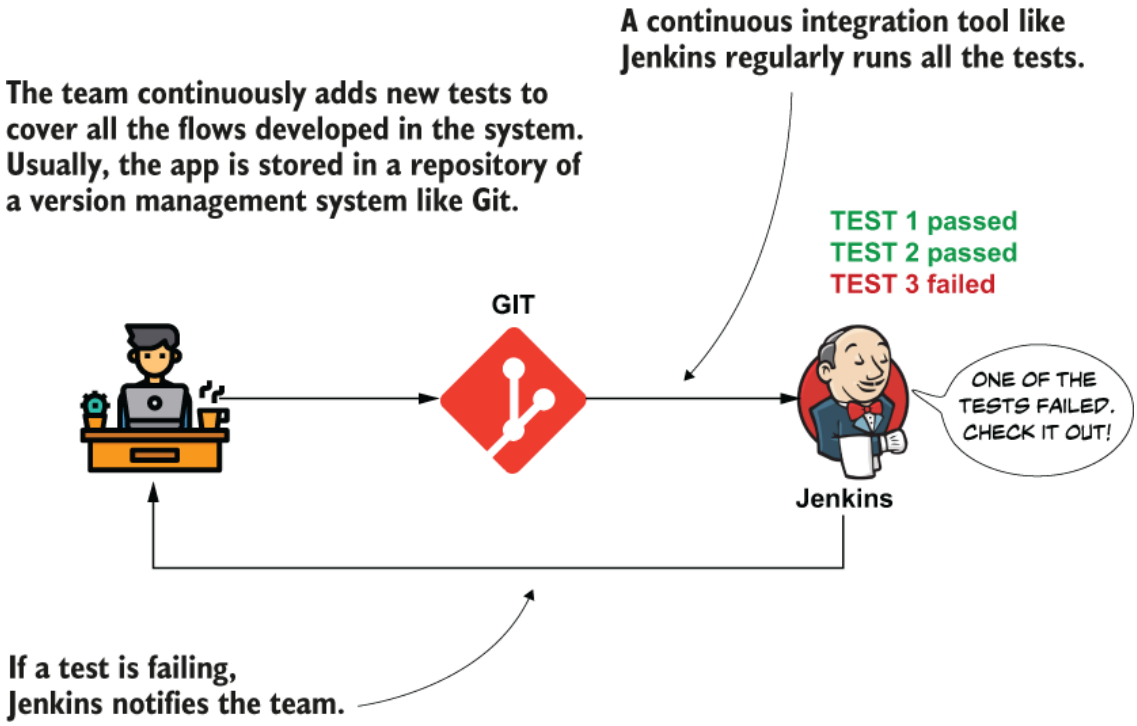

Figure 1.7 The team deploys the testing app in a test environment. A continuous integration tool like Jenkins executes the app regularly and sends feedback to the team. This way, the team is always aware of the system's status, and they know if they break something during development.

Such an application might become as complex as a backend app. In order to validate the flows, the app needs to communicate with the components of the system and even connect to databases. Sometimes the app mocks external dependencies to simulate different execution scenarios. For writing the test scenarios, developers use frameworks like Selenium, Cucumber, Gauge, and others. But, together with these frameworks, the app could still benefit in several ways from Spring's tools. For example, the app could manage the object instances to make the code more maintainable using the Spring IoC container. It could use Spring Data to connect to the databases where it needs to validate the data. It could send messages to queues or topics of a broker system to simulate specific scenarios or simply use Spring to call some REST endpoints (figure 1.8). (Remember, it's okay if this looks too advanced; meaning will be clarified as you progress through the book).

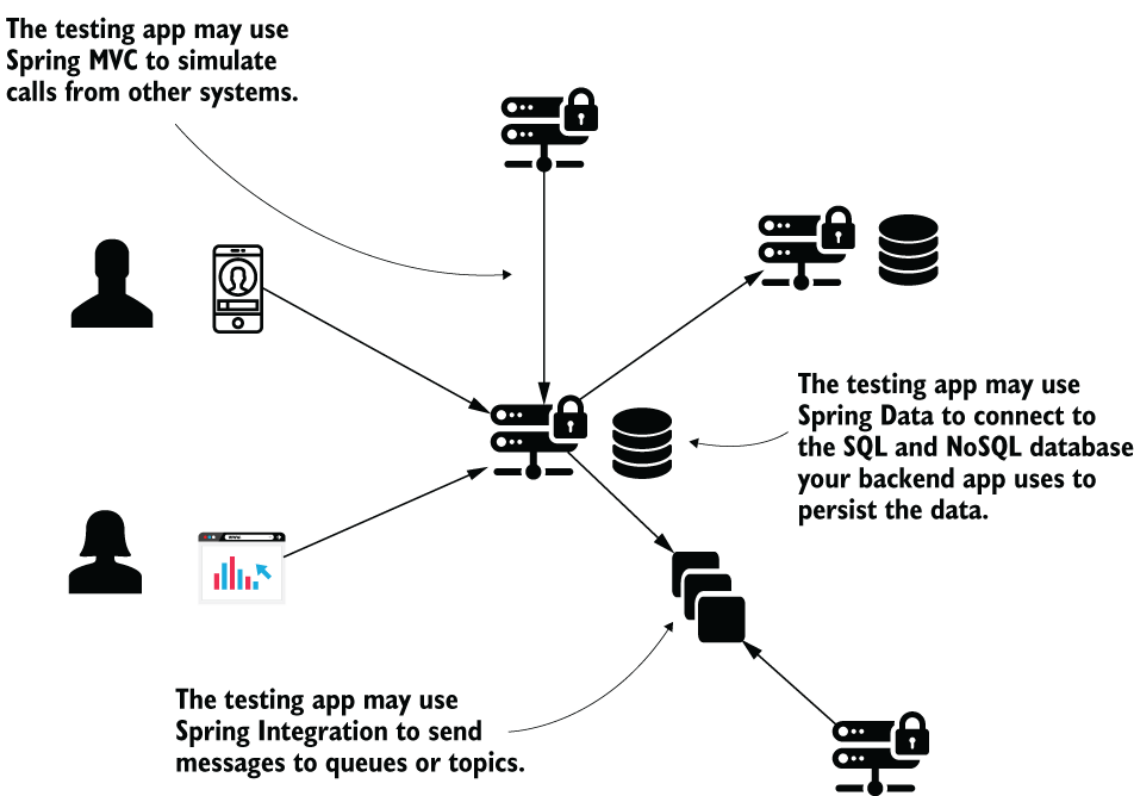

Figure 1.8 A testing app might need to connect to databases or communicate with other systems or the tested system. The developers can use components of the Spring ecosystem to simplify the implementations of these functionalities.

1.3.3 Using Spring for the development of a desktop app

Today, desktop applications are not that frequently developed, as web or mobile apps have taken the role of interacting with the user. However, there's still a small number of desktop applications, and components of the Spring ecosystem could be a good choice in the development of their features. A desktop app could successfully use the Spring IoC container to manage the object instances. This way, the app's implementation is cleaner and improves its maintainability. Additionally, the app could potentially use Spring's tools to implement different features, for example to communicate with a backend or other components (calling web services or using other techniques for remote calls) or implement a caching solution.

1.3.4 Using Spring in mobile apps

With its Spring for Android project (**https://spring.io/projects/spring-android**), the Spring community tries to help the development of mobile applications. Even though you'll probably rarely encounter this situation, it's worth mentioning that you can use Spring's tools to develop Android

apps. This Spring project provides a REST client for Android and authentication support for accessing secured APIs.

1.4 When not to use frameworks

In this section, we discuss why you should sometimes avoid using frameworks. It's essential you know when to use a framework and when to avoid using them. Sometimes, using a tool that's too much for the job might consume more energy and also obtain a worse result. Imagine using a chainsaw to cut bread. While you could try to and even achieve a final result, it'd be more difficult and energy-consuming than using a regular knife (and you may end up with nothing but breadcrumbs instead of sliced bread). We'll discuss a few scenarios in which using a framework isn't a great idea, and then I'll tell you a story about a team I was part of that failed in the implementation of an app because of using a framework.

It turns out that, like everything else in software development, you shouldn't apply a framework in all cases. You'll find situations in which a framework is not a good fit—or maybe a framework is a good fit, but not the Spring framework. In which of the following scenarios should you consider not using a framework?

1. You need to implement a particular functionality with a footprint as small as possible. By footprint, I mean the storage memory occupied by the app's files.
2. Specific security requirements force you to implement only custom code in your app without making use of any open source framework.
3. You'd have to make so many customizations over the framework that you'd write more code than if you'd simply not used it at all.
4. You already have a functional app, and by changing it to use a framework you don't gain any benefit.

Let's discuss these points in more detail.

1.4.1 You need to have a small footprint

For point one, I refer to situations in which you need to make your application small. In today's systems, we find more and more cases in which the services are delivered in containers. You've likely heard about con-

tainers, such as Docker, Kubernetes, or other terms related to this subject (if not, again, that's okay).

Containers in their entirety is a topic beyond the scope of this book, so for now the only thing I need you to know is that when you use such a deployment fashion, you want your application to be as small as possible. A container is like a box in which your application lives. One crucial principle regarding app deployment in containers is that the containers should be easily disposable: they can be destroyed and recreated as fast as possible. The size of the app (footprint) matters a lot here. You can save seconds from the app initialization by making it smaller. That doesn't mean you won't use frameworks for all the apps deployed in containers.

But for some apps, which are usually also quite small, it makes more sense to improve their initialization and make their footprint smaller rather than adding dependencies to different frameworks. Such a case is a kind of application called *server-less function*. These server-less functions are tiny applications deployed in containers. Because you don't have too much access to the way they're deployed, it looks like they execute without a server (hence their name). These apps need to be small, and that's why, for this specific case of apps, you'll want to avoid adding a framework as much as possible. Because of its size, it's also possible that you won't need a framework anyway.

1.4.2 Security needs dictate custom code

I said in point two that in specific situations, apps could not use frameworks because of security requirements. This scenario usually happens with apps in the field of defense or governmental organizations. Again, it doesn't mean all the apps used in governmental organizations are prohibited from using frameworks, but for some, restrictions are applied. You may wonder why. Well, say an open source framework like Spring is used. If someone finds a specific vulnerability, it will become known, and a hacker could use this knowledge to exploit it. Sometimes, stakeholders of such apps want to make sure the chances of someone hacking into their system is as close to zero as possible. This could lead to even rebuilding a functionality instead of using it from a third-party source.

NOTE Wait! Earlier I said that it's more secure to use an open source framework because if a vulnerability exists, someone will likely discover

it. Well, if you invest enough time and money, you probably can achieve this yourself as well. In general, it's cheaper to use a framework, of course. And if you don't want to be extra cautious, it makes more sense to use a framework. But in some projects, the stakeholders really want to make sure no information becomes public.

1.4.3 Abundant existing customizations make a framework impractical

Another case (point three) in which you might want to avoid using a framework is when you'd have to customize its components so much that you end up writing more code than if it hadn't been used. As I specified in section 1.1, a framework provides you parts that you assemble with your business code to obtain an app. These components, provided by the framework, don't fit perfectly, and you need to customize them in different ways. It's perfectly normal to customize the framework's components and the style in which they assemble than if you'd developed the functionality from scratch. If you find yourself in such a situation, you have probably chosen the wrong framework (search for alternatives) or you shouldn't use a framework at all.

1.4.4 You won't benefit from switching to a framework

In point four, I mentioned that a potential mistake could be trying to use a framework to replace something that already exists and is working in an app. Sometimes we are tempted to replace an existing architecture with something new. A new framework appears, and it's popular, and everyone uses it, so why shouldn't we change our app as well to use this framework? You can, but you need to attentively analyze what you want to achieve by changing something that works. In some cases, like my story from section 1.1, it could be helpful to change your app and make it rely on a specific framework. As long as this change brings a benefit, do it! A reason could be that you want to make the app more maintainable, more performant, or more secure. But if this change doesn't bring you a benefit, and sometimes it might even bring incertitude, then, in the end, you might discover you invested the time and money for a worse result. Let me tell you a story from my own experience.

1.5 What will you learn in this book

Since you opened this book, I assume you're probably a software developer in the Java ecosystem who found out it's useful to learn Spring. The purpose of this book is to teach you the foundations of Spring, assuming you know nothing at all about frameworks and, of course, about Spring. When I say Spring, I refer to the Spring ecosystem, not just the core part of the framework.

When you finish the book, you will have learned how to do the following:

- Use the Spring context and implement aspects around objects managed by the framework.
- Implement the mechanism of a Spring app to connect to a database and work with the persisted data.
- Establish data exchange between apps using REST APIs implemented with Spring.
- Build basic apps that use the convention-over-configuration approach.
- Use best practices in the standard class design of a Spring application.
- Properly test your Spring implementations.

An avoidable mistake

Using frameworks isn't always the best choice, and I had to learn that the hard way. Years earlier, we were working on the backend of a web application. Times influence many things, including software architectures. The app was using JDBC to directly connect to an Oracle database. The code was quite ugly. Everywhere the app needed to execute a query on the database it opened a statement and then sent a query that was sometimes written on multiple rows. You might be young enough not to have encountered JDBC direct usage in apps, but trust me, it's a long and ugly code.

At that time, some frameworks using another methodology to work with the database were becoming more and more popular. I remember when I first encountered Hibernate. This is an ORM framework, which allows you to treat the tables and their relationships in a database as objects and relationships among objects. When used correctly, it enables you to write

less code and more intuitive functionality. When misused, it may slow down your app, make the code less intuitive, and even introduce bugs.

The application we were developing needed a change. We knew we could improve that ugly JDBC code. In my mind, we could at least minimize the number of lines. This change would have brought great benefits to maintainability. Together with other developers, we suggested using a tool provided by Spring called `JdbcTemplate` (you'll learn this tool in chapter 12). But others strongly pushed the decision to use Hibernate. It was quite popular, so why not to use it? (Actually it still is one of the most popular frameworks of its kind, and you'll learn about integrating it with Spring in chapter 13.) I could see changing that code to a completely new methodology would be a challenge. Moreover, I could see no benefits. The change also implied a greater risk of introducing bugs.

Fortunately, the change started with a proof of concept. After a couple of months, lots of effort, and stress, the team decided to quit.

After analyzing our options, we finished the implementation using `JdbcTemplate`. We managed to write cleaner code by eliminating a large number of lines of code, and we didn't need to introduce any new framework for this change.

Summary

- An application framework is a set of common software functionalities that provides a foundational structure for developing an application. A framework acts as the skeletal support to build an application.
- A framework helps you build an app more efficiently by providing functionality that you assemble to your implementation instead of developing it yourself. Using a framework saves you time and helps ensure there are fewer chances of implementing buggy features.

- Using a widely known framework like Spring opens a door to a large community, which makes it more likely that others will faces similar problems. You then have an excellent opportunity to learn about how others solved something similar to an issue you need to address, which will spare you the time of individual research.
- When implementing an application, always think of all possibilities, including not using a framework. If you decide to use one or more frameworks, take into consideration all their alternatives. You should think about the purpose of the framework, who else is using it (how big the community is), and for how long it's been on the market (maturity).
- Spring is not just a framework. We often refer to Spring as "Spring framework" to indicate the core functionalities, but Spring offers an entire ecosystem formed of many projects used in application development. Each project is dedicated to a specific domain, and when implementing an app, you might use more of these projects to implement the functionality you desire. The projects of the Spring ecosystem we'll use in this book are as follows:
    - Spring Core, which builds the foundation of Spring and provides features like the context, aspects, and basic data access.
    - Spring Data, which provides a high-level, comfortable-to-use set of tools to implement the persistence layer of your apps. You'll find how easy it is to use Spring Data to work with both SQL and NoSQL databases.
    - Spring Boot, which is a project of the Spring ecosystem that helps you apply a "convention-over-configuration" approach.
- Quite often, learning materials (like books, articles, or video tutorials) offer examples with Spring only for backend applications. While it's true that it's widespread to use Spring with backend apps, you can use Spring with other kinds of apps as well, even in desktop applications and automation testing apps.