When I graduated from Northern Illinois University back in 2001 after spending most of the previous two years working on COBOL, mainframe Assembler, and Job Control Language (JCL), I took a job as a consultant to learn Java. I specifically took that position because of the opportunity to learn Java when it was the hot new thing. Never in my wildest dreams did I think I’d be back writing about batch processing. I’m sure most Java developers don’t think about batch, either. They think about the latest web framework or JVM language. They think about service-oriented architectures and things like REST versus SOAP or whatever alphabet soup is hot at the time.

But the fact is, the business world runs on batch. Your bank and 401k statements are all generated via batch processes. The e-mails you receive from your favorite stores with coupons in them? Probably sent via batch processes. Even the order in which the repair guy comes to your house to fix your laundry machine is determined by batch processing. In a time when we get our news from Twitter, Google thinks that waiting for a page refresh takes too long to provide search results, and YouTube can make someone a household name overnight, why do we need batch processing at all?

There are a few good reasons:

- You don’t always have all the required information immediately. Batch processing allows you to collect information required for a given process before starting the required processing. Take your monthly bank statement as an example. Does it make sense to generate the file format for your printed statement after every transaction? It makes more sense to wait until the end of the month and look back at a vetted list of transactions from which to build the statement.

- Sometimes it makes good business sense. Although most people would love to have what they buy online put on a delivery truck the second they click Buy, that may not be the best course of action for the retailer. If a customer changes their mind and wants to cancel an order, it’s much cheaper to cancel if it hasn’t shipped yet. Giving the customer a few extra hours and batching the shipping together can save the retailer large amounts of money.

- It can be a better use of resources. Having a lot of processing power sitting idle is expensive. It’s more cost effective to have a collection of scheduled processes that run one after the other using the machine’s full potential at a constant, predictable rate.

This book is about batch processing with the framework Spring Batch. This chapter looks at the history of batch processing, calls out the challenges in developing batch jobs, makes a case for developing batch using Java and Spring Batch, and finally provides a high-level overview of the framework and its features.
CHAPTER 1 ■ BATCH AND SPRING

A History of Batch Processing

To look at the history of batch processing, you really need to look at the history of computing itself. The time was 1951. The UNIVAC became the first commercially produced computer. Prior to this point, computers were each unique, custom-built machines designed for a specific function (for example, in 1946 the military commissioned a computer to calculate the trajectories of artillery shells). The UNIVAC consisted of 5,200 vacuum tubes, weighed in at over 14 tons, had a blazing speed of 2.25MHz (compared to the iPhone 4, which has a 1GHz processor) and ran programs that were loaded from tape drives. Pretty fast for its day, the UNIVAC was considered the first commercially available batch processor.

Before going any further into history, I should define what, exactly, batch processing is. Most of the applications you develop have an aspect of user interaction, whether it’s a user clicking a link in a web app, typing information into a form on a thick client, or tapping around on phone and tablet apps. Batch processing is the exact opposite of those types of applications. Batch processing, for this book’s purposes, is defined as the processing of data without interaction or interruption. Once started, a batch process runs to some form of completion without any intervention.

Four years passed in the evolution of computers and data processing before the next big change: high-level languages. They were first introduced with Lisp and Fortran on the IBM 704, but it was the Common Business Oriented Language (COBOL) that has since become the 800-pound gorilla in the batch-processing world. Developed in 1959 and revised in 1968, 1974, and 1985, COBOL still runs batch processing in modern business. A Gartner study estimated that 60% of all global code and 85% of global business data is housed in the language. To put this in perspective, if you printed out all that code and stacked the printout, you’d have a stack 227 miles high. But that’s where the innovation stalled.

COBOL hasn’t seen a significant revision in a quarter of a century. The number of schools that teach COBOL and its related technologies has declined significantly in favor of newer technologies like Java and .NET. The hardware is expensive, and resources are becoming scarce.

Mainframe computers aren’t the only places that batch processing occurs. Those e-mails I mentioned previously are sent via batch processes that probably aren’t run on mainframes. And the download of data from the point-of-sale terminal at your favorite fast food chain is batch, too. But there is a significant difference between the batch processes you find on a mainframe and those typically written for other environments (C++ and UNIX, for example). Each of those batch processes is custom developed, and they have very little in common. Since the takeover by COBOL, there has been very little in the way of new tools or techniques. Yes, cron jobs have kicked off custom-developed processes on UNIX servers and scheduled tasks on Microsoft Windows servers, but there have been no new industry-accepted tools for doing batch processes.

Until now. In 2007, Accenture announced that it was partnering with Interface21 (the original authors of the Spring framework, and now SpringSource) to develop an open source framework that would be used to create enterprise batch processes. As Accenture’s first formal foray into the open source world, it chose to combine its expertise in batch processing with Spring’s popularity and feature set to create a robust, easy-to-use framework. At the end of March 2008, the Spring Batch 1.0.0 release was made available to the public; it represented the first standards-based approach to batch processing in the Java world. Slightly more than a year later, in April 2009, Spring Batch went 2.0.0, adding features like replacing support for JDK 1.4 with JDK 1.5+, chunk-based processing, improved configuration options, and significant additions to the scalability options within the framework.

1 http://www.gartner.com/webletter/merant/article1/article1.html
2 There have been revisions in COBOL 2002 and Object Oriented COBOL, but their adoption has been significantly less than for previous versions.