***Templating data access***

You’ve probably traveled by plane before. If so, you’ll surely agree that one of the most important parts of traveling is getting your luggage from point A to point B. There are many steps to this process: When you arrive at the terminal, your first stop is at the counter to check your luggage. Next, security scans it to ensure the safety of the flight. Then it takes a ride on the luggage train on its way to being placed on the plane. If you need to catch a connecting flight, your luggage needs to be moved, as well. When you arrive at your final destination, the luggage has to be removed from the plane and placed on the carousel. Finally, you go down to the baggage claim area and pick it up. Even though there are many steps to this process, you’re actively involved in only a couple of them. The carrier is responsible for driving the process. You’re involved only when you need to be; the rest is taken care of. This mirrors a powerful design pattern: the template method pattern.

A template method defines the skeleton of a process. In the example, the process is moving luggage from departure city to arrival city. The process itself is fixed; it never changes. The overall sequence of events for handling luggage occurs the same way every time: luggage is checked in, luggage is loaded onto the plane, and so forth. Some steps of the process are fixed as well—they happen the same way every time. When the plane arrives at its destination, every piece of luggage is unloaded one at a time and placed on a carousel to be taken to baggage claim.

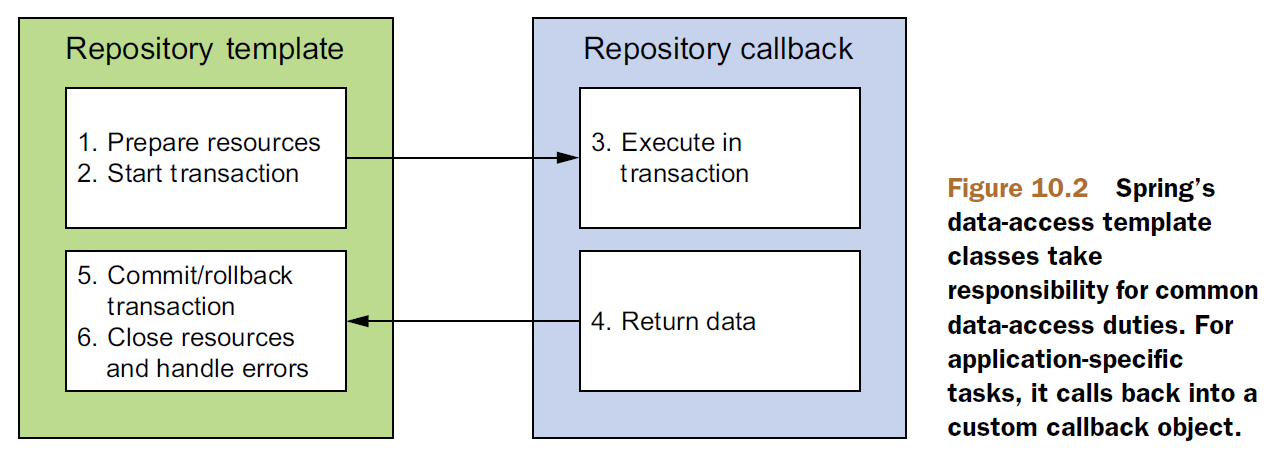
At certain points, the process delegates its work to a subclass to fill in some implementation specific details. This is the variable part of the process. For example, the handling of luggage starts with a passenger checking in the luggage at the counter. This part of the process always has to happen at the beginning, so its sequence in the

process is fixed. Because each passenger’s luggage check-in is different, the implementation of this part of the process is determined by the passenger. In software terms, a template method delegates the implementation-specific portions of the process to an interface. Different implementations of this interface define specific implementations of this portion of the process.

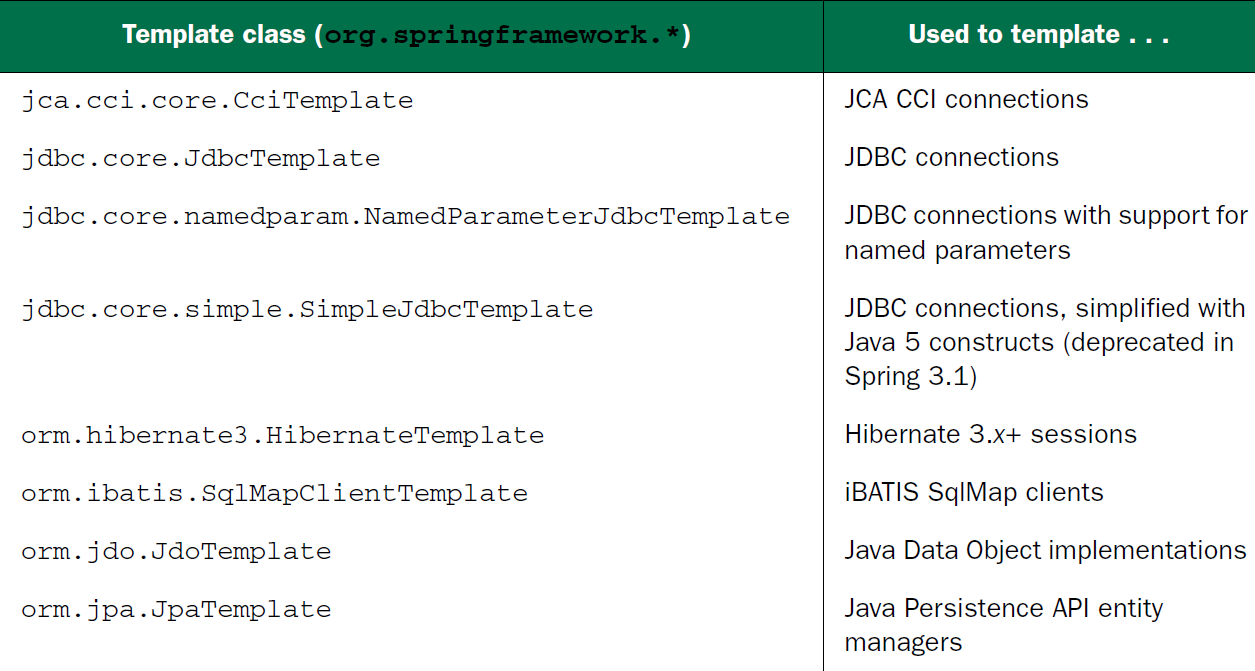
This is the same pattern that Spring applies to data access. No matter what technology you’re using, certain data-access steps are required. For example, you always need to obtain a connection to your data store and clean up resources when you’re done. These are the fixed steps in a data-access process. But each data-access method you

write is slightly different. You query for different objects and update the data in different ways. These are the variable steps in the data-access process.

* **Spring separates the fixed and variable parts of the data-access process into two distinct classes: *templates* and *callbacks*.** Templates manage the fixed part of the process, whereas your custom data-access code is handled in callbacks.

****

* Spring’s template classes handle the fixed parts of data access—controlling transactions, managing resources, and handling exceptions.
* Meanwhile, the specifics of data access as they pertain to your application—creating statements, binding parameters, and marshaling result sets—are handled in the callback implementation.
* Spring comes with several templates to choose from, depending on your persistence platform choice. If you’re using straight JDBC, then you’ll want to use JdbcTemplate. But if you favor one of the object-relational mapping frameworks, perhaps HibernateTemplate or JpaTemplate is more suitable.

****