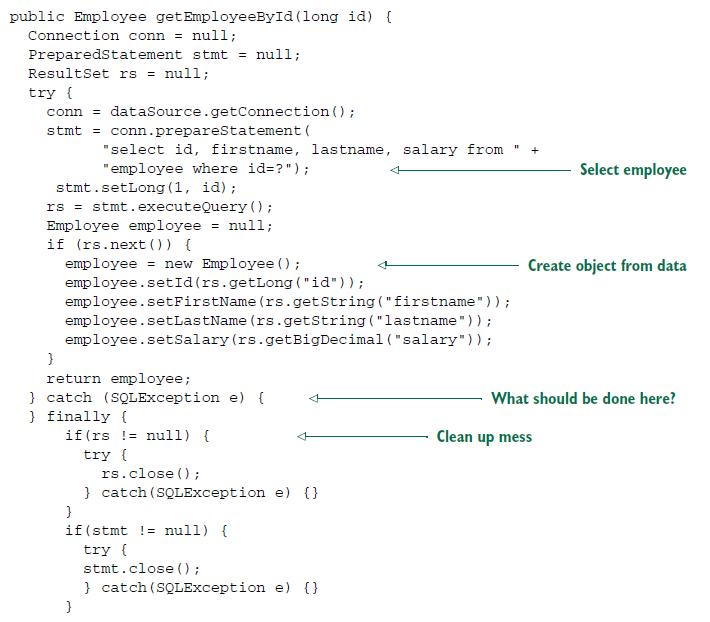
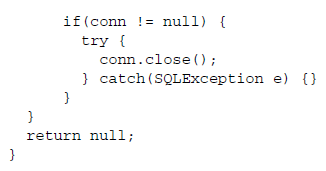
***Eliminating boilerplate code with templates***

* Have you ever written a code and then felt like you’d already written the same code before?

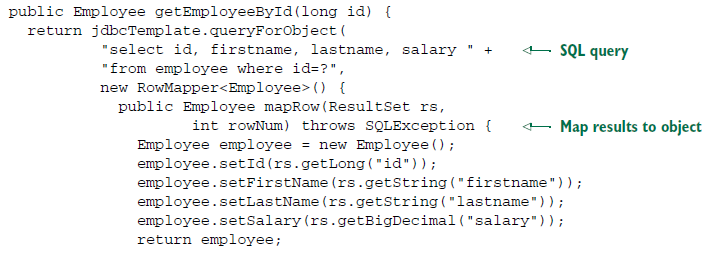
That’s *boilerplate code—* the code that you often have to write over and over again to accomplish common and otherwise simple tasks.

* Unfortunately, there are a lot of places where Java APIs involve a bunch of boiler plate code. A common example of boilerplate code can be seen when working with JDBC to query data from a database.





* As you can see with JDBC code queries the database for an employee’s name and salary. But I’ll bet you had to look hard to see that. That’s because the small bit of code that’s specific to querying for an employee is buried in a heap of JDBC ceremony. You first have to create a connection, then create a statement, and finally query for the results. And, to appease JDBC’s anger, you must catch *SQLException*, a checked exception, even though there’s not a lot you can do if it’s thrown. Finally, after all is said and done, you have to clean up the mess , closing down the connection, statement , and result set . This could also stir JDBC’s anger, so you must catch *SQLException* here as well.
* JDBC is not alone in the boilerplate code business. Many activities require similar boilerplate code. JMS, JNDI, and the consumption of REST services often involve a lot of commonly repeated code.
* Sprint seeks to eliminate boilerplate code by encapsulating it in templates.
* Spring’s JdbcTemplate makes it possible to perform database operations without all the ceremony required by traditional JDBC.
* For example, using Spring’s SimpleJdbcTemplate (a specialization of JdbcTemplate that takes advantage of Java 5 features) , the *getEmployeeById()*  method cab be rewritten so that its focus is on the task of retrieving employee data and not catering to the demands of the JDBC API. The following shows what such an updated *getEmployeeById()* method might look like.





* As you can see, this new version of *getEmployeeById()* is much simpler and acutely focused on selecting an employee from the database. The template’s *queryForObject()* method is given the SQL query, a *RowMapper* (for mapping result set data to a domain object), and zero or more query parameters. What you don’t see in *getEmployeeById()* is any of the JDBC boilerplate from before. Everything is handled inside the template.
* We have seen how Spring attacks complexity in Java development using POJO oriented development, DI, aspects, and templates. Along the way, I showed you how to configure beans and aspects in XML-based configuration files. But how do those files get loaded? And what are they loaded into? Let’s look at the Sprint Container, the place where your application’s beans reside.