Flyway's SQL-script based database migration is more than powerful enough for most use cases. But sometimes, you need to take it a step further to adapt your existing data to the new database schema. In these cases, Flyway's Java migration and callback methods provide an easy and powerful way to implement the necessary migration logic.

Implement Complex Migrations in Java

Similar to the SQL migration steps, Flyway automatically finds and executes the required migration classes. You just have to implement the *JdbcMigration* interface, add your class to the package defined in Flyway's location property and use a class name that follows Flyway's naming schema *V<VERSION>__DESCRIPTION.java*.

Here's an example of a class which updates the database to version 2.0.

```
public class V2__my_migration implements JdbcMigration {


    @Override
    public void migrate(Connection connection) throws
Exception {
        // implement your migration here …
    }
}
```

Use Callbacks for Repetitive Tasks

Another useful feature for complex migration scenarios is Flyway's callback mechanism. It allows you to execute an SQL script or a Java class when one of the following lifecycle events gets triggered within Flyway:

- beforeMigrate
- beforeEachMigrate

- afterEachMigrate
- afterMigrate
- beforeClean
- afterClean
- beforeInfo
- afterInfo
- beforeValidate
- afterValidate
- beforeBaseline
- afterBaseline
- beforeRepair
- afterRepair

SQL Callbacks

The implementation of an SQL callback is straightforward. You just need to add an SQL script, with the name of the lifecycle trigger you want to use, to your migration directory. The migration directory is either the sql folder of the Flyway command line client or the *src/main/resources/db/migration* folder of your Java application.

So, if you want to execute a SQL script after Flyway migrated your database, you just need to put all SQL statements into a file with the name *afterMigrate.sql* and copy it to the *sql* or *src/main/resources/db/migration* folder.

Java Callbacks

If your callback operation is too complex for an SQL script, you can also implement it in Java. That requires 2 steps. You need a class that implements the *FlywayCallback* interface and register it in your *Flyway* instance.

If you need access to the Flyway configuration, you should also implement the *ConfigurationAware* interface. It defines a method which Flyway will call with the configuration object so that you can store it in an internal property and use it in your callback implementation.

Another class you should know is the *BaseFlywayCallback* class. It implements the *FlywayCallback* and the *ConfigurationAware* interface and provides empty implementations for all methods.

So, you just need to override the callback methods you want to implement, like in the following class which only provides an implementation for the afterMigrate lifecycle callback.

```
public class MyAfterMigrateCallback extends
BaseFlywayCallback {

    @Override

    public void afterMigrate(Connection connection) {

        // implement your callback here …

    }

}
```

You now just need to register the callback implementation in your Flyway instance. You do that by calling the *setCallbacks* method of the *Flyway* class with one or more *FlywayCallback* implementations.

```
Flyway flyway = new Flyway();

flyway.setDataSource("jdbc:postgresql://localhost:5432/test_f
lyway", "postgres", "postgres");

flyway.setCallbacks(new MyAfterMigrateCallback());

flyway.migrate();
```