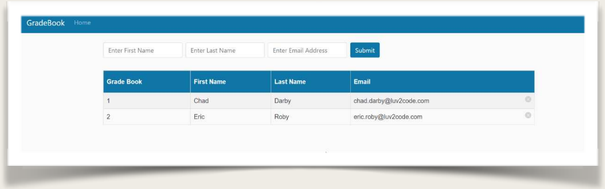
Course Project - Testing Overview

Student Grade Book App

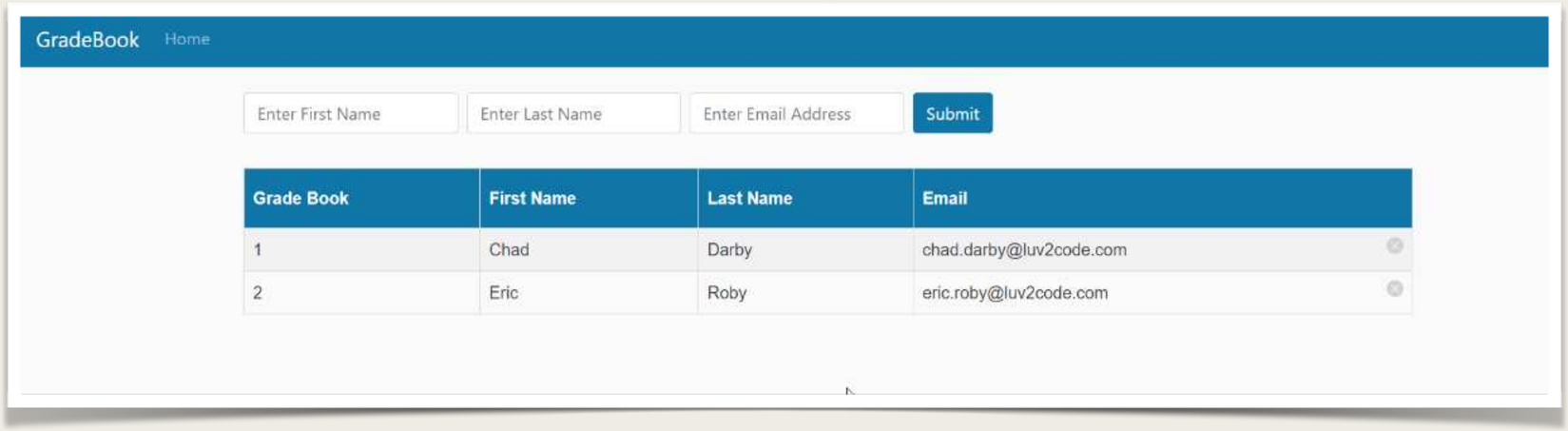
- We will start with an existing Student Grade Book App
- The app was created by a previous employee ... but it is unfinished (yikes!)



- Our job:
  - Add remaining functionality to save data in database
  - Add unit tests and integration tests

About Student Grade Book App

- An instructor can keep track of grades for a student
- Grades are tracked for the subjects: History, Science and Math
- Instructor can add grades for a student for a specific subject



Technical Stack

- Spring Boot
- Spring Data JPA
- Spring MVC
- Thymeleaf views
- CSS and JavaScript

DEMO

Existing Code

Controller

```
GradeBookController.java
```

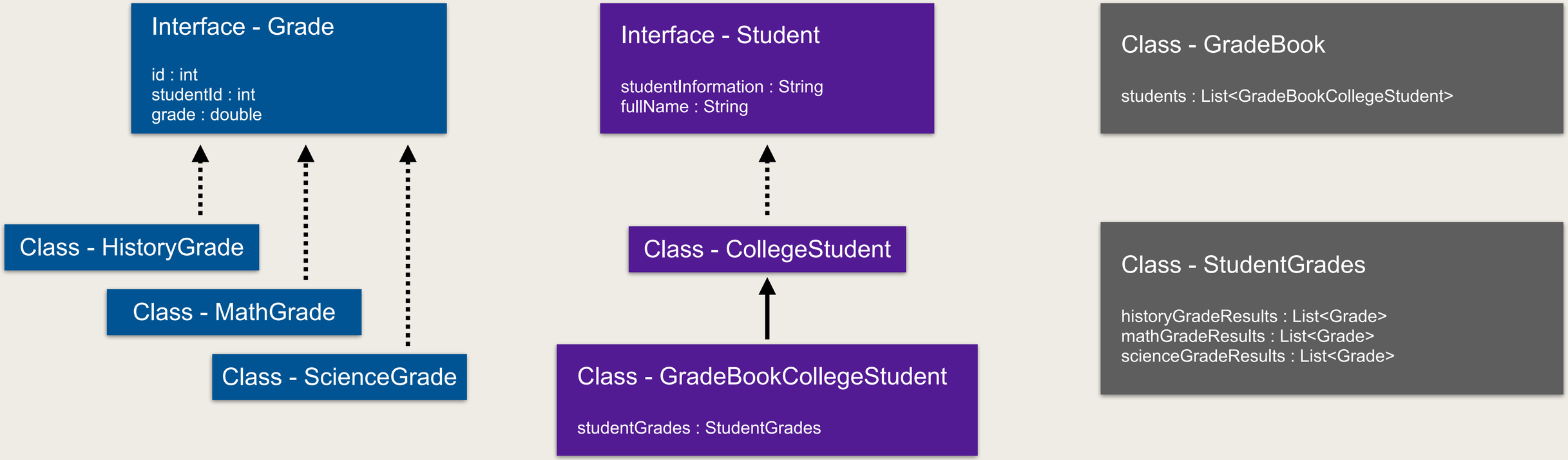
View

```
index.html
studentInformation.html
error.html
cssandjs/
```

Model

```
CollegeStudent.java
Grade.java
Gradebook.java
GradebookCollegeStudent.java
HistoryGrade.java
MathGrade.java
ScienceGrade.java
Student.java
StudentGrades.java
```

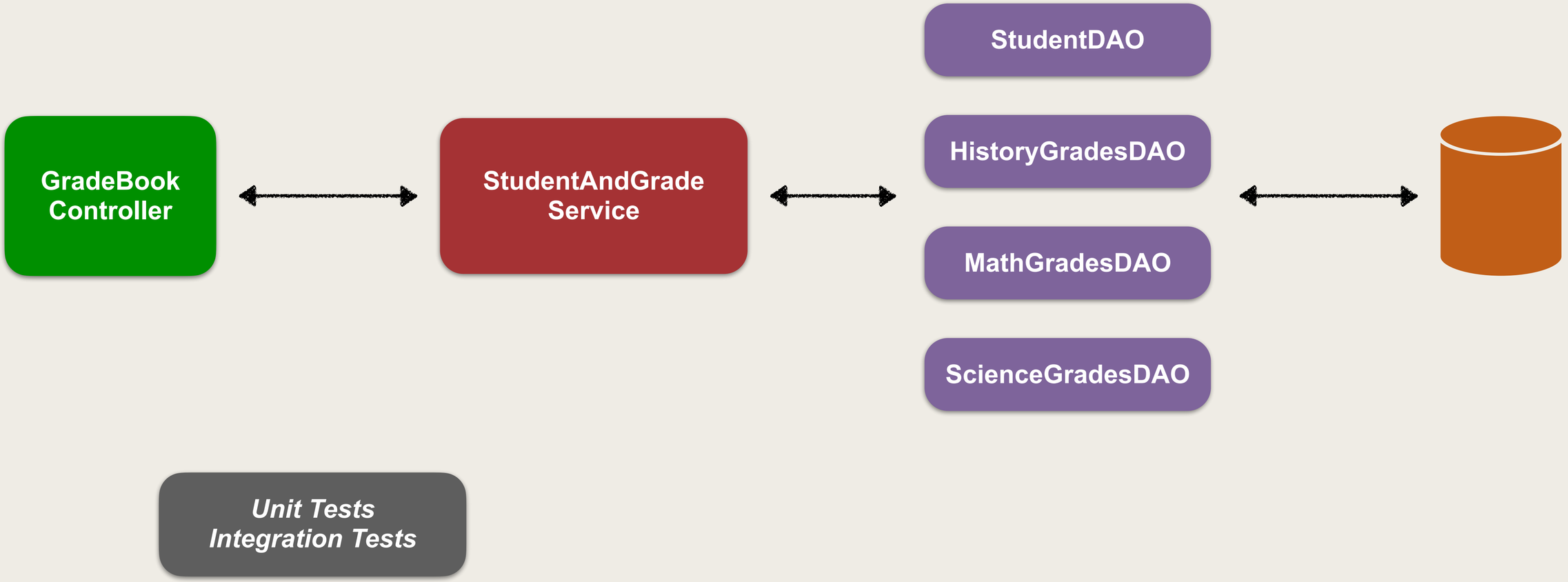
Existing Code - Model Classes



Code we will develop

- Currently, the app does not store information in database
- We'll add DAO database support
- We'll also add a service class
- During development, add unit tests and integration tests

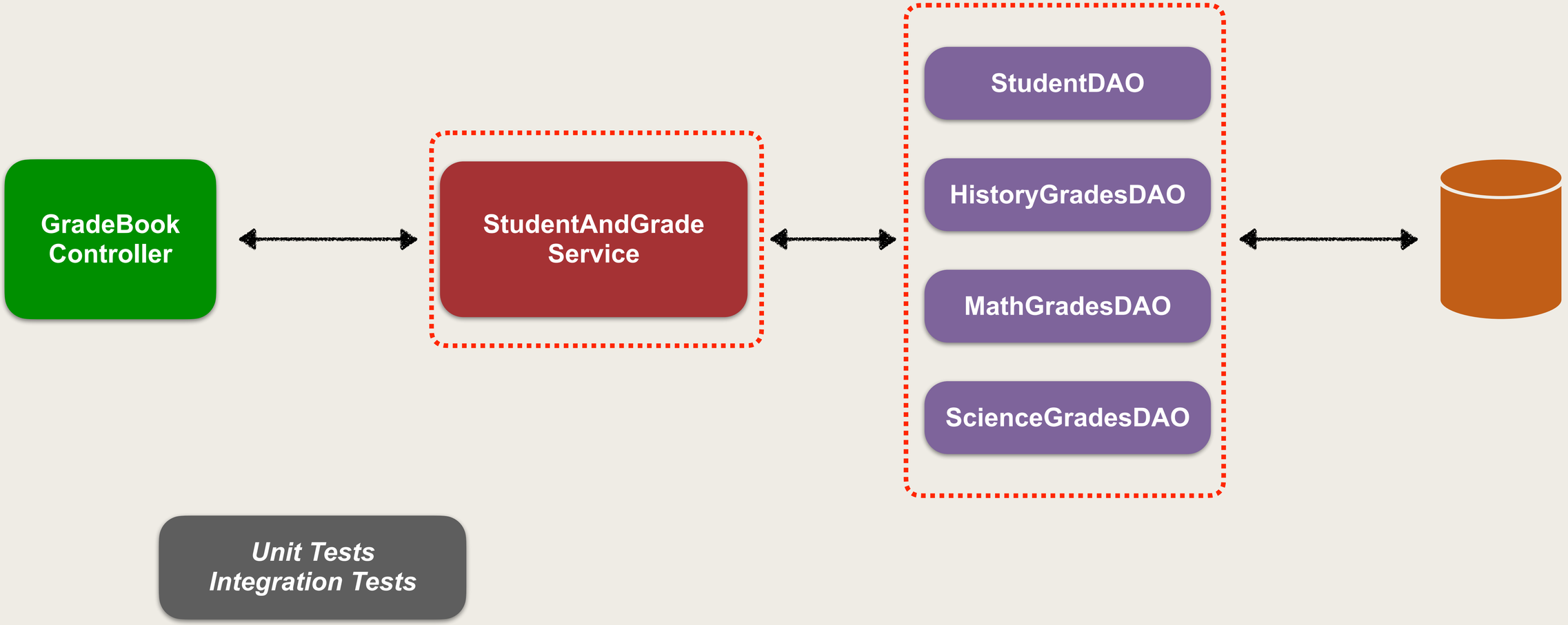
Final Architecture



luv2code

TDD for Service and DAO

Use TDD to Build Service and DAOs



DAOs and DB

- For DAOs, we will make use of Spring Data JPA
- For database, we will use H2 database (in-memory, embedded db)
  - In-memory, embedded db is good for testing
  - Quickly set up and tear down
  - No network latency so tests run faster
  - Minimizes left over data in the database

luv2code

Database Integration Testing

Database Initialization and Cleanup

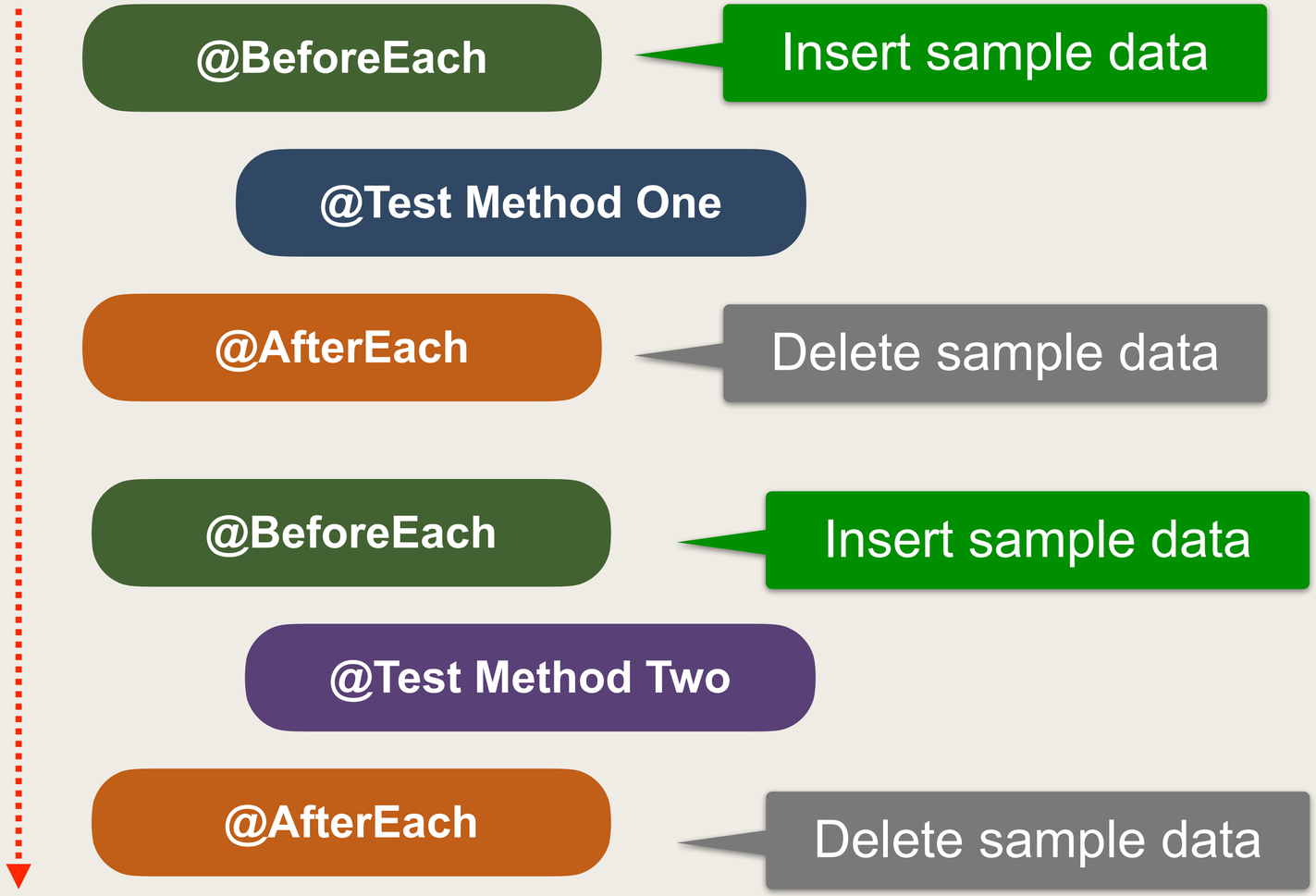
- When we are performing integration testing with a database
  - Each test should run from a known state

- Before each test, perform initialization
  - Insert sample data

- After each test, perform cleanup
  - Delete the sample data

Testing Approach

- Each test should run from a known state



@Before and @AfterEach

StudentAndGradeServiceTest.java

```
import org.springframework.jdbc.core.JdbcTemplate;
import org.junit.jupiter.api.AfterEach;
import org.junit.jupiter.api.BeforeEach;
...

@TestPropertySource("/application.properties")
@SpringBootTest
public class StudentAndGradeServiceTest {

   @Autowired
   private JdbcTemplate jdbc;

   @BeforeEach
   public void setupDatabase() {
       jdbc.execute("insert into student(id, firstname, lastname, email_address) " +
               "values (1, 'Eric', 'Roby', 'eric.roby@luv2code_school.com')");
   }


   @AfterEach
   public void setupAfterTransaction() {
       jdbc.execute("DELETE FROM student");
   }

}
```

From the Spring Framework

Insert sample data

Delete sample data

StudentAndGradeServiceTest.java

```
...
public class StudentAndGradeServiceTest {

   @Autowired
   private JdbcTemplate jdbc;

   @BeforeEach
   public void setupDatabase() {
       jdbc.execute("insert into student(id, firstname, lastname, email_address) " +
               "values (1, 'Eric', 'Roby', 'eric.roby@luv2code_school.com')");
   }

   @Test
   public void isStudentNullCheck() {

       assertTrue(studentService.checkIfStudentIsNull(1));

       assertFalse(studentService.checkIfStudentIsNull(0));
   }

   @AfterEach
   public void setupAfterTransaction() {
       jdbc.execute("DELETE FROM student");
   }

}
```

Returns true since
id 1 exists in database

Returns false since
id 0 does not exist in database