In this chapter, you will learn about basic testing techniques you can apply in testing Java applications, and also the testing support features offered by the Spring framework. These features can make your testing tasks easier and lead you to better application design. In general, applications developed with the Spring framework and the dependency injection pattern are easy to test.

Testing is a key activity in software development for ensuring quality. There are many types of testing, including unit testing, integration testing, functional testing, system testing, performance testing, and acceptance testing. Spring’s testing support focuses on unit and integration testing, but it can also help with other types of testing. Testing can be performed either manually or automatically. However, since automated tests can be run repeatedly and continuously at different phases of a development process, they are highly recommended, especially in agile development processes. The Spring framework is an agile framework that fits these kinds of processes.

There are many testing frameworks available on the Java platform. Currently, JUnit and TestNG are the most popular. JUnit has a long history and a large user group in the Java community. JUnit had major improvements from version 3.8 to 4.0, including support for annotations. TestNG is another popular Java testing framework that makes extensive use of annotations. Compared to JUnit, TestNG offers additional powerful features such as test grouping, dependent test methods, and data-driven tests.

In releases prior to 2.5, Spring offered testing support specific to JUnit 3.8, which is referred to as JUnit 3.8 legacy support. You can take advantage of this support by extending the base test classes it provides. Spring 2.5 upgrades its testing support features by offering the Spring TestContext framework, which requires Java 1.5 or higher. This framework abstracts the underlying testing framework with the following concepts:

**Test context:** This encapsulates the context of a test’s execution, including the application context, test class, current test instance, current test method, and current test exception.

**Test context manager:** This manages a test context for a test and triggers test execution listeners at predefined test execution points, including when preparing a test instance, before executing a test method (before any framework-specific initialization methods), and after executing a test method (after any framework-specific cleanup methods).

**Test execution listener:** This defines a listener interface, by implementing which you can listen to test execution events. The TestContext framework provides several test execution listeners for common testing features, but you are free to create your own.
Spring 2.5 provides convenient TestContext support classes for JUnit 3.8, JUnit 4.4, and TestNG 5.5, with particular test execution listeners preregistered. You can simply extend these support classes to use the TestContext framework without having to know much about the framework details.

Upon finishing this chapter, you will understand the basic concepts and techniques of testing and the popular Java testing frameworks JUnit and TestNG. You will also be able to create unit tests and integration tests using both the JUnit 3.8 legacy support and the Spring TestContext framework.

12-1. Creating Tests with JUnit and TestNG

Problem

You would like to create automated tests for your Java application so that they can be run repeatedly to ensure the correctness of your application.

Solution

The most popular testing frameworks on the Java platform are JUnit and TestNG. JUnit 4 incorporates several major improvements over JUnit 3.8, which relies on the base class (i.e., TestCase) and the method signature (i.e., methods whose names begin with test) to identify test cases—an approach that lacks flexibility. JUnit 4 allows you to annotate your test methods with JUnit's @Test annotation, so an arbitrary public method can be run as a test case. TestNG is another powerful testing framework that makes use of annotations. It also provides a @Test annotation type for you to identify test cases.

How It Works

Suppose you are going to develop a system for a bank. To ensure the system's quality, you have to test every part of it. First, let's consider an interest calculator, whose interface is defined as follows:

```java
package com.apress.springrecipes.bank;

public interface InterestCalculator {
    public void setRate(double rate);
    public double calculate(double amount, double year);
}
```

Each interest calculator requires a fixed interest rate to be set. Now you can implement this calculator with a simple interest formula:

```java
package com.apress.springrecipes.bank;

public class SimpleInterestCalculator implements InterestCalculator {
    private double rate;

    public SimpleInterestCalculator() {
        this.rate = 0.05;
    }

    public void setRate(double rate) {
        this.rate = rate;
    }

    public double calculate(double amount, double year) {
        return amount * (1 + rate) * year;
    }
}
```