Testing the Behavior

> *"Do not allow watching food to replace making food."*
>
> *--Alton Brown*

How important is it to clearly state your intended actions? When driving a car on an empty street at night, do you use a turning signal to let any unseen pedestrians know what you intend to do? It is too easy to write a test that seems fine, but after two months of working on something else, it looks completely cryptic and incomprehensible.

In this chapter, we will be converting our tests from a cryptic set of method calls into something that any person off the street can understand. The actions of the test will remain the same, but how each action is described will become dramatically clearer. In this chapter, we will cover the following topics:

- Behavior-driven Development
- The write once, test everywhere pattern
- JBehave
- Cucumber

Behavior-driven Development

Writing a test that clearly states its intent, is useful in itself. However, as we get into the habit of making ourselves clearly understood, we start to notice a pattern. **Behavior-driven Development** (**BDD**) encourages us to step back and think of how the application should behave end-to-end first, and only then concentrate on the smaller details. After all, our application can be refactored many times with all of the IDs and names changing while still maintaining the same intended behavior.

Advantages of BDD

By separating the implementation details from the behavior definition, our tests gain a lot of advantages:

- **Better test understanding**: If the test is written properly, then it is possible to know exactly what the test plans to do without being confused by code details.
- **Modular implementation**: The methods that perform the actual implementation can be shared while testing.

NOTE

For more information on code reuse, see the *The DRY testing pattern* section in Chapter 3, *Refactoring Tests*.

- **Versatile implementation options**: By sticking closely to behavior, it is easy to have one defined behavior running in multiple environments. This will further be explained in the write once, test everywhere pattern later in this chapter.
- **Multiple BDD frameworks**: There are multiple testing frameworks written to allow you to test with the BDD principle in just about every programming language.
- **Data separation**: Data used by the test is extracted out of the behavior definition, making it easier to manage the data in the long run.

NOTE

To study further, refer to the *Hardcoding input data* section in Chapter 4, *Data-driven Testing*.

Disadvantages of BDD

There are several disadvantages of using BDD tools; some teams might find that the negatives outweigh the positives. Here are some examples of the disadvantages of BDD:

- **Consistent specification language**: If you ask 10 people to describe a spoon in one sentence, you will get 10 different sentences. Having the whole team agree on how a registration flow should be described in a consistent plain language can be a nightmare. Without having a consistent standard, it is easy to create duplicate code based on how someone wishes to describe an action. For example, `I click on the product link` and `I follow the link to product page` could be describing the same method call.

NOTE

Gojko Adzic describes ways to bridge the communication gap between team members in *Specification by Example: How Successful Teams Deliver the Right Software*, *Manning Publications*.

- **Easy to mix behavior and implementation**: It is very temptingly easy to start adding implementation into a definition. This practice leads to muddled, confused, and hardcoded tests.
- **Which BDD tool to use**: Any team might have a long and heated debate over which tool is perfect for a project. Choosing the perfect tool might be very difficult.
- **Added overhead**: Adding another framework to a project makes writing tests simpler. However, each tool will use precious resources such as time or processing power.

- **Learning curve**: Each new framework will have a learning curve before everyone on the team can use the tools effectively.