The above `CustomerCollection` class is derived from `Collection<Customer>` because we don't want to derive it from `List<>` for the reasons mentioned earlier. Now we are passing a `List<>` in the default constructor because we might need to call some useful methods in `List<T>` and that's why we are encapsulating `List<T>` like this, deriving our `Collection<T>` using the base constructor so that the `Customer Collection` can simply call the underlying `List<T>`'s find (or any other useful method) implementation, saving us from having to implement our own.

Summary

We have seen how a 5-tier architecture works. An important point to note is that there can be multiple ways of implementing a 5-tier architecture, and no architectural implementation can be a silver bullet for your own custom project. The aim of this chapter is to give you an idea of and an approach to how a 5-tier design can be implemented in your own projects. But this is by no means the only best method to do so. Many experienced developers implement their own custom n-tier solutions, which can be very different from what we have seen in this chapter. But the basic concepts from an architectural standpoint would be similar to what we have learnt in this chapter, giving us the knowledge to customize our own implementation for our specific project needs.

We also covered some crucial architectural aspects and design patterns, such as loose coupling, a lack of strong dependency on other layers, scalability, and so on. An n-tier implementation involves more work and longer code files, but this is a small price to pay for greater benefits such as scalability, maintainability and flexibility, which are important for big projects.

It's important to understand that we should not blindly use an n-tier architecture in every project. We need to think about and foresee a need for such distributed architecture and then plan accordingly. Using a 5-tier architecture for a simple guestbook application for a personal website would be overkill, in addition to wasting time and resources. But it is a must for a large-scale inventory management system that needs to interact with other external systems such as accounting packages. With proper use, n-tier architecture can help organizations scale up without re-writing the entire application, and can save valuable man hours of work in the long run, as well as provide a stable and robust application platform for future growth.