Building Configuration Hierarchy and Overriding

In this lesson, we will learn to build levels of configuration hierarchy.

WE'LL COVER THE FOLLOWING

Configuration hierarchy overview

- The framework should bundle the default value for all the configurable parameters in a configuration file.
- These default values should be overridable from a project-specific configuration file.
- These project-specific configuration parameters should be overridable by passing as a JVM argument (i.e., `-Dkey=value`).

Designing the configuration manager

- The reading of configuration files needs to happen only once. This can be achieved by using the Singleton Pattern for the creation of the file object.
- The configuration file can be any of these formats: `.properties`, `.ini`, `.xml`, `.json`, `.yaml`, `.toml`.
- We can use an appropriate library for reading the configuration files for fetching the configuration parameters.

Configuration overriding

First, the configuration will be looked up in the JVM arguments, so it can be passed from the command line as shown below:

```
-Ddatabase.enabled=true
```

If the configuration is not passed through the command line, it will be looked up in the project-specific configuration file.

`project-specific-config.properties`

```
database.enabled = true
```

If the configuration parameter is not passed through the command line or a project-specific file, then it will fallback to the default configuration file.

`default-config.properties`

```
database.enabled = false
```

The below `ConfigurationManager` class is a sample implementation demonstrating configuration hierarchy overriding. For demonstration, we have used `.properties` file format.

```
import java.io.IOException;
import java.util.Properties;

public class ConfigurationManager {

    private ConfigurationManager() throws IOException {
        PROPERTIES.load(ConfigurationManager.class.getResourceAsStream("de
fault-config.properties"));
        PROPERTIES.load(ConfigurationManager.class.getResourceAsStream("pr
oject-specific-config.properties"));
    }

    private static ConfigurationManager manager;

    private static final Properties PROPERTIES = new Properties();

    public static ConfigurationManager getInstance() {

        if (manager == null) {
            synchronized (ConfigurationManager.class) {
                if (manager == null) {
                    try {
                        manager = new ConfigurationManager();
```

```
                    } catch (IOException e) {
                        e.printStackTrace();
                    }
                }
            }
        }
        return manager;
    }

    public String getProperty(String name) {
        return System.getProperty(name, PROPERTIES.getProperty(name));
    }
}
```

The code below reads the configuration parameter's value, and the overriding happens in the `ConfigurationManager` class while fetching the value.

```
boolean isDatabaseEnabled = "true".equalsIgnoreCase(ConfigurationManager.g
etInstance().getProperty("database.enabled"));
```

Now that you are familiar with the configuration hierarchy, in the next lesson, we will learn about logging.