tutorialspoint

SIMPLY EASY LEARNING

www.tutorialspoint.com

About the Tutorial

Guice is an open source, Java-based dependency injection framework. It is quiet lightweight and is actively developed/managed by Google. This tutorial covers the concepts required for a basic understanding of Google Guice and gives you a feel of how it works.

This tutorial gives you a comprehensive coverage of concepts of Guice and makes you comfortable to use it in your software development projects.

Audience

This tutorial has been prepared for the beginners to help them understand the basic to advanced concepts related to Google Guice.

Prerequisites

Before you start practicing various types of examples given in this reference, we assume that you are already aware about computer programs and computer programming languages.

Copyright & Disclaimer

Table of Contents

1. Guice – Overview

Guice is an open source, Java-based dependency injection framework. It is lightweight and is developed as well as managed by Google. This chapter will give you an overview of Guice framework.

Dependency Injection

Any Java-based application contains objects that work together to present what the end-user sees as a final working application. When writing a complex Java application, application classes should be independent of other Java classes as much possible to increase their reusability. This also makes them independent of other classes while unit testing. Dependency Injection, also termed as wiring, helps in gluing these classes together and at the same time keeping them independent.

Consider you have an application which has a text editor component and you want to provide a spell check. Your standard code would look something like this –

```
public class TextEditor {
   private SpellChecker spellChecker;
   public TextEditor() {
      spellChecker = new SpellChecker();
   }
}
```

Note that here we have created a dependency between the TextEditor and the SpellChecker. In an inversion of control scenario, we would instead do something like this –

```
public class TextEditor {
   private SpellChecker spellChecker;
      @Inject
   public TextEditor(SpellChecker spellChecker) {
      this.spellChecker = spellChecker;
   }
}
```

Here, the TextEditor is not concerned about SpellChecker implementation. The SpellChecker will be implemented independently and will be provided to the TextEditor at the time of TextEditor instantiation.

Dependency Injection using Guice (Binding)

Dependency Injection is controlled by the Guice Bindings. Guice uses bindings to map object types to their actual implementations. A module is a collection of bindings as shown below –

```
public class TextEditorModule extends AbstractModule {
   @Override
   protected void configure() {
      /*
         * Bind SpellChecker binding to WinWordSpellChecker implementation
         * whenever spellChecker dependency is used.
      */
      bind(SpellChecker.class).to(WinWordSpellChecker.class);
   }
}
```

An injector is the object-graph builder and a Module is its core building block. Thus, the first step is to create an injector and then use the injector to get the objects.

```
public static void main(String[] args) {
   /*
      * Guice.createInjector() takes Modules, and returns a new Injector
      * instance. This method is to be called once during application startup.
   */

   Injector injector = Guice.createInjector(new TextEditorModule());
   /*
      * Build object using injector
   */
   TextEditor textEditor = injector.getInstance(TextEditor.class);
}
```

In the above example, TextEditor class object graph is constructed by Guice and this graph contains TextEditor object and its dependency as WinWordSpellChecker object.

2. Guice – Environmental Setup

Let us look into setting up the environment for Guice, before proceeding into its functionalities.

Local Environment Setup

This section guides you on how to download and set up Java environment on your machine. Please follow the steps mentioned below to set up the environment.

Java SE is freely available from the link Download Java. Download a version based on your operating system. Then, follow the instructions to download Java and run the **.exe** file to install Java on your machine. Once you installed Java, you would need to set environment variables to point to correct installation directories.

Setting up the Path for Windows 2000/XP

We are assuming that you have installed Java in *c:\Program Files\java\jdk*directory –

- Right-click 'My Computer' and select 'Properties'.
- Click the 'Environment variables' button under the 'Advanced' tab.
- Now, alter the 'Path' variable so that it also contains the path to the Java executable. For example, if the path is currently set to **'C:\WINDOWS\SYSTEM32'**, then change your path to read **'C:\WINDOWS\SYSTEM32;c:\Program Files\java\jdk\bin'**.

Setting up the Path for Windows 95/98/ME

In case, you have installed Java in **c:\Program Files\java\jdkdirectory**, edit the **'C:\autoexec.bat'** file and add the following line at the end: **'SET PATH=%PATH%;C:\Program Files\java\jdk\bin'**.

Setting up the Path for Linux, UNIX, Solaris, FreeBSD

Environment variable PATH should be set to point to where the Java binaries have been installed. You may refer to your shell documentation for more information.

For example, if you use **bash** as your shell, then you would add the following line to the end: '**.bashrc: export PATH=/path/to/java:$PATH'**

Popular Java Editors

To write your Java programs, you need a text editor. There are many sophisticated IDEs available in the market. But for now, you can consider one of the following –

- **Notepad**: On Windows, you can use any simple text editor like Notepad (Recommended for this tutorial), or TextPad.
- **Netbeans**: It is a Java IDE that is open-source and free, which can be downloaded from https://www.netbeans.org/index.html.
- **Eclipse**: It is also a Java IDE developed by the Eclipse open-source community and can be downloaded from https://www.eclipse.org/.

Google Guice Environment

Download the latest version of Google Guice and related jar files.

- Google Guice 4.0
- AOP Alliance 1.0
- Guava 16.0.1
- javax.inject 1.0

Note that in this tutorial, we have copied all these into **C:\>Google** folder.

OS	Archive name
Windows	guice-4.1.0.jar;aopalliance-1.0.jar;guava-16.0.1.jar;javax.inject-1.jar
Linux	guice-4.1.0.jar;aopalliance-1.0.jar;guava-16.0.1.jar;javax.inject-1.jar
Mac	guice-4.1.0.jar;aopalliance-1.0.jar;guava-16.0.1.jar;javax.inject-1.jar

Set CLASSPATH Variable

Set the **CLASSPATH** environment variable to point to the Guice jar location, assuming you have stored Guice and the related jars in Google folder on various Operating Systems as follows.

OS	Output
Windows	Set the environment variable CLASSPATH to %CLASSPATH%;C:\Google\guice-4.1.0.jar;C:\Google\aopalliance-1.0.jar;C:\Google\guava-16.0.1.jar;C:\Google\javax.inject-1.jar;.;
Linux	export CLASSPATH=$CLASSPATH:Google/guice-4.1.0.jar:Google/aopalliance-1.0.jar:Google/guava-16.0.1.jar:Google/javax.inject-1.jar:.
Mac	export CLASSPATH=$CLASSPATH:Google/guice-4.1.0.jar:Google/aopalliance-1.0.jar:Google/guava-16.0.1.jar:Google/javax.inject-1.jar:.

3. Google Guice – First Application

In this chapter, let us look into a sample console based application where well demonstrate dependency injection using Guice binding mechanism step by step, with their respective code.

Step 1: Create Interface

```
//spell checker interface
interface SpellChecker {
   public void checkSpelling();
}
```

Step 2: Create Implementation

```
//spell checker implementation
class SpellCheckerImpl implements SpellChecker {
   @Override
   public void checkSpelling() {
      System.out.println("Inside checkSpelling." );
   }
}
```

Step 3: Create Bindings Module

```
//Binding Module
class TextEditorModule extends AbstractModule {
   @Override
   protected void configure() {
      bind(SpellChecker.class).to(SpellCheckerImpl.class);
   }
}
```

Step 4: Create Class with dependency

```
class TextEditor {
   private SpellChecker spellChecker;

   @Inject
   public TextEditor(SpellChecker spellChecker) {
```

```
      this.spellChecker = spellChecker;
   }
   public void makeSpellCheck() {
      spellChecker.checkSpelling();
   }
}
```

Step 5: Create Injector

```
Injector injector = Guice.createInjector(new TextEditorModule());
```

Step 6: Get Object with dependency fulfilled

```
TextEditor editor = injector.getInstance(TextEditor.class);
```

Step 7: Use the object

```
editor.makeSpellCheck();
```

Complete Example

Create a java class named GuiceTester.

GuiceTester.java

```
import com.google.inject.AbstractModule;
import com.google.inject.Guice;
import com.google.inject.Inject;
import com.google.inject.Injector;

public class GuiceTester {
   public static void main(String[] args) {
      Injector injector = Guice.createInjector(new TextEditorModule());
      TextEditor editor = injector.getInstance(TextEditor.class);
      editor.makeSpellCheck();
   }
}
class TextEditor {
   private SpellChecker spellChecker;
  @Inject

   public TextEditor(SpellChecker spellChecker) {
      this.spellChecker = spellChecker;
```

```
   }
   public void makeSpellCheck() {
      spellChecker.checkSpelling();
   }
}

//Binding Module
class TextEditorModule extends AbstractModule {
   @Override

   protected void configure() {
      bind(SpellChecker.class).to(SpellCheckerImpl.class);
   }
}

//spell checker interface
interface SpellChecker {
   public void checkSpelling();
}

//spell checker implementation
class SpellCheckerImpl implements SpellChecker {
   @Override

   public void checkSpelling() {
      System.out.println("Inside checkSpelling." );
   }
}
```

Output

Now, compile and run the file. You can see the following output:

```
Inside checkSpelling.
```

==========