|  |  |
| --- | --- |
| Introduction to Google Guice. Introduction | by Mohit Sharma | Medium  Google Guice  SQE Project | Abstract  A Google Guice Report. Guice is an open source, Java-based dependency injection framework. It is lightweight and is actively developed/managed by Google.  Sean Daly & Andrew Lennox  Software Development |

Contents

[*What is Google Guice?* 5](#_Toc68010431)

[*What is Dependency Injection?* 5](#_Toc68010432)

[*Dependency Injection* 6](#_Toc68010433)

[*Dependency Injection using Guice (Binding)* 7](#_Toc68010434)

[*Local Environment Setup* 8](#_Toc68010435)

[*Google Guice Environment* 8](#_Toc68010437)

[*Google Guice – Sample Application* 8](#_Toc68010438)

[*Step 1: Create Interface* 8](#_Toc68010440)

[*Step 2: Create Implementation* 8](#_Toc68010441)

[*Step 3: Create Bindings Module* 9](#_Toc68010442)

[*Step 4: Create Class with dependency* 9](#_Toc68010443)

[*Step 5: Create Injector* 9](#_Toc68010444)

[*Step 6: Get Object with dependency fulfilled* 9](#_Toc68010445)

[*Step 7: Use the object* 9](#_Toc68010446)

[*Complete Example* 10](#_Toc68010447)

[*Output* 10](#_Toc68010448)

[*Google Guice - Linked Binding* 11](#_Toc68010449)

[*Complete Example* 12](#_Toc68010450)

[*Output* 12](#_Toc68010451)

[*Google Guice – Binding Annotations* 13](#_Toc68010452)

[*Create a binding annotation* 13](#_Toc68010453)

[*Mapping using binding annotation* 13](#_Toc68010454)

[*Inject using binding annotation* 13](#_Toc68010455)

[*Complete Example* 14](#_Toc68010456)

[*Output* 15](#_Toc68010457)

[*Google Guice - @Named Binding* 16](#_Toc68010458)

[*Mapping using named annotation* 16](#_Toc68010459)

[*Inject using @Named annotation* 16](#_Toc68010460)

[*Complete Example* 17](#_Toc68010461)

[*Output* 18](#_Toc68010462)

[*Google Guice – Constant Bindings* 19](#_Toc68010463)

[*Complete Example* 20](#_Toc68010466)

[*Output* 21](#_Toc68010467)

[*Google Guice - Provider Class* 22](#_Toc68010468)

[*Complete Example* 23](#_Toc68010469)

[*Output* 24](#_Toc68010470)

[*Google Guice - Constructor Bindings* 25](#_Toc68010471)

[*Complete Example* 26](#_Toc68010472)

[*Output* 27](#_Toc68010473)

[*Google Guice - Inbuilt Bindings* 27](#_Toc68010474)

[*Complete Example* 27](#_Toc68010475)

[*Output* 27](#_Toc68010476)

[*Google Guice - Constructor Injection* 28](#_Toc68010477)

[*Output* 29](#_Toc68010478)

[*Google Guice - Method Injection* 30](#_Toc68010479)

[*Output* 31](#_Toc68010480)

[*Google Guice - Field Injection* 32](#_Toc68010481)

[*Output* 33](#_Toc68010483)

[*Google Guice - Optional Injection* 34](#_Toc68010484)

[*Output* 35](#_Toc68010485)

### **What is Google Guice?**

Guice is a Java-based dependency injection application that is open source. It's very light and Google is actively developing/managing it.

This guide covers the majority of the topics needed to gain a clear understanding of Google Guice and how it works. Guice is a Google project that allows you to build a graph of dependencies so that you can create complex structures out of simpler pieces.

Guice is a method that's used in a lot of Java development in a more concrete sense. Beyond that, it has a positive effect on the code, reducing boilerplate and increasing testability.

### **What is Dependency Injection?**

Dependency injection is a software engineering technique in which an object receives other objects from which it relies. Dependencies are the names given to these other objects. The receiving object in a standard "using" relationship is referred to as a client, while the transferred (or "injected") object is referred to as a service. The injector is the code that transfers the service to the client and can be any number of items. Rather than the client deciding which service it will use, the injector instructs the client. The term "injection" refers to a dependency (a service) being passed into the entity (a client) that will use it.

The client's state is integrated into the operation. The pattern's fundamental requirement is that the service be passed to the client rather than allowing the client to construct or locate the service.

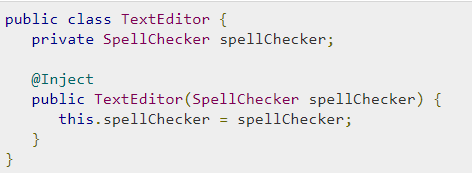
The aim of dependency injection is to separate the concerns about object creation and usage. This can help with code reuse and readability.

Dependency injection is a subset of the inversion of control strategy. A customer does not have to know how to build those programs in order to use them. Instead, the client assigned the role of delivering services to third-party code (the injector).

After that, the injector injects (passes) the services into the client, which may already exist or be installed by the injector. After that, the customer makes use of the facilities. This means the client doesn't need to know about the injector, how the services are designed, or even which services it's using. The client just needs to be aware of the services' intrinsic interfaces, which determine how the client can use the services. This distinguishes between the "usage" and "construction" obligations.

# Dependency Injection

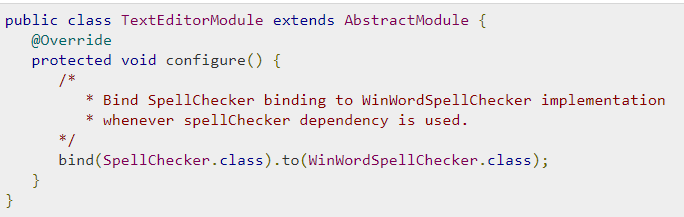
Consider the following scenario: you have a text editor component in your application and you want to provide a spell check. Your typical code would be something like this –

What we've done here is, We've created a dependency between the TextEditor and the SpellChecker here. Instead, in an inversion of control scenario, we would do something like this –

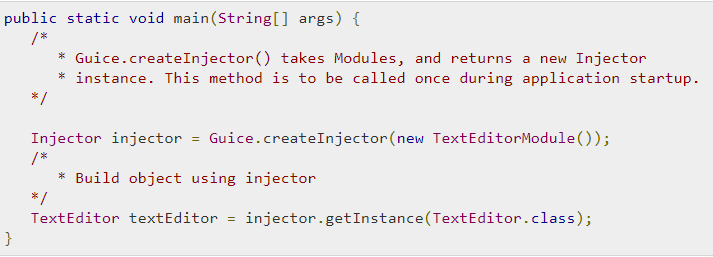
The TextEditor should not be concerned about SpellChecker implementation in this case. The SpellChecker will be implemented independently and made available to the TextEditor at the time the TextEditor is instantiated.

# Dependency Injection using Guice (Binding)

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



The Module is the core building block for an Injector which is Guice's object-graph builder. First step is to create an injector and then we can use the injector to get the objects.



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

# Local Environment Setup

Java SE can be downloaded for free from the Download Java link. As a result, you download the version that corresponds to your operating system.

# Follow the on-screen instructions to download Java and run the.exe to install it on your computer. After you've installed Java on your machine, you'll need to configure environment variables to point to the proper installation directories.

# Google Guice Environment

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

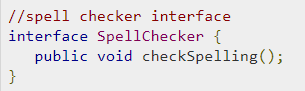
* [Google Guice 4.0](https://mvnrepository.com/artifact/com.google.inject/guice/4.0)
* [AOP Alliance 1.0](https://mvnrepository.com/artifact/aopalliance/aopalliance/1.0)
* [Guava 16.0.1](https://mvnrepository.com/artifact/com.google.guava/guava/16.0.1)
* [Javax.Inject 1.0](https://mvnrepository.com/artifact/javax.inject/javax.inject/1)

Set the CLASSPATH environment variable to point to the Guice jar location.

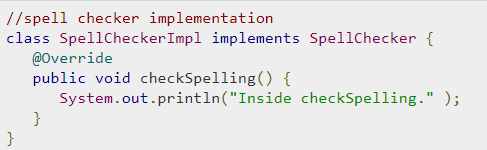
# Google Guice – Sample Application

## Let's build a simple console application to demonstrate dependency injection using the Guice binding mechanism step by step. Bindings specify how Guice will inject dependencies into a class.

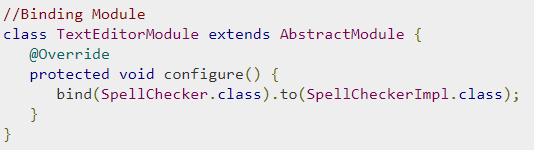
## Step 1: Create Interface



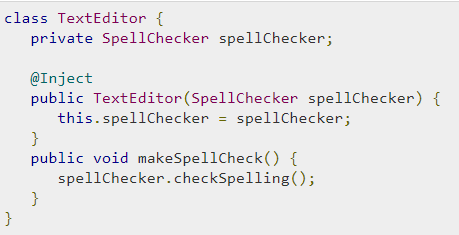
## Step 2: Create Implementation



## Step 3: Create Bindings Module



## Step 4: Create Class with dependency



## Step 5: Create Injector



## Step 6: Get Object with dependency fulfilled



## Step 7: Use the object



## Complete Example



## Output



# Google Guice - Linked Binding

Guice maps a type to its implementation in Linked bindings. We've mapped the SpellChecker interface to its implementation SpellCheckerImpl in the example below.

We can also map the concrete class to its subclass.



## Complete Example



## Output



# Google Guice – Binding Annotations

We can bind a type with its implementation. In case we want to map a type with multiple implementations, we can create custom annotation as well.

## Create a binding annotation

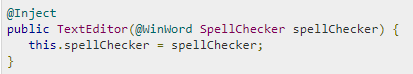


* **@BindingAnnotation** − Marks annotation as binding annotation.
* **@Target** − Marks’s applicability of annotation.
* **@Retention** − Marks’s availability of annotation as runtime.

## Mapping using binding annotation

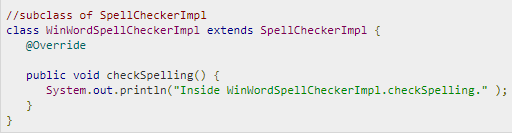


## Inject using binding annotation



## Complete Example





## Output



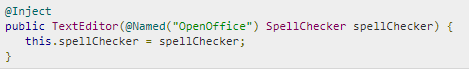
# Google Guice - @Named Binding

Guice provides another way to map bindings without creating custom annotation by using @Named annotation.

## Mapping using named annotation



## Inject using @Named annotation



## Complete Example



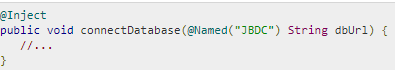
## Output



# Google Guice – Constant Bindings

## Guice allows you to make bindings with value objects or constants. In this example, we will consider the situation where we need to configure a JDBC URL.

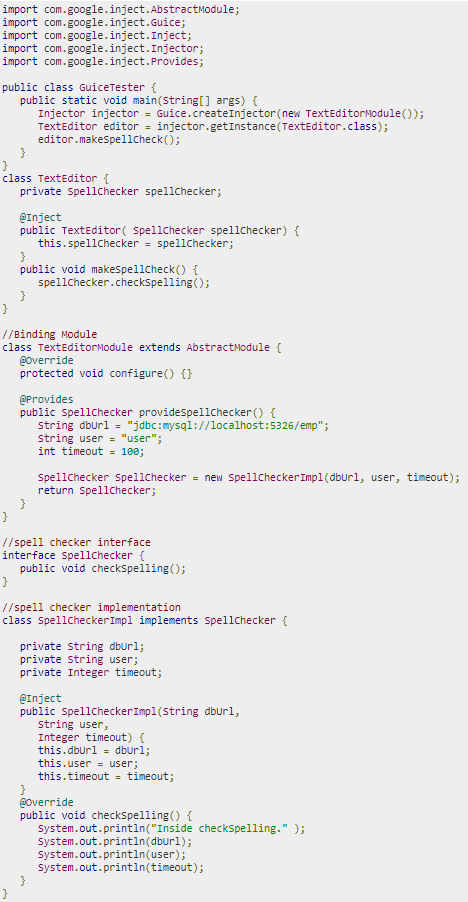
## Inject using @Named annotation



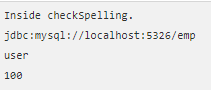
This can be achived using toInstance() method.



## Complete Example

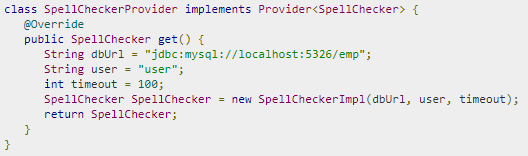


## Output



# Google Guice - Provider Class

As @provides method becomes more complex, these methods can be moved to separate classes using a Provider interface.

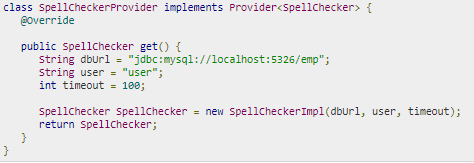


Next step is to map the provider to type.

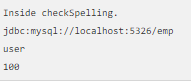


## Complete Example



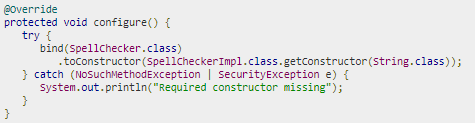


## Output



# Google Guice - Constructor Bindings

Guice allows us to make bindings with particular constructors of an object utilizing toConstructor() method.



## Complete Example



## Output



# Google Guice - Inbuilt Bindings

Guice offers inbuilt binding for java.util.logging.Logger class. Logger's name is automatically set to the name of the class into which the Logger is injected.

## Complete Example



## Output



# Google Guice - Constructor Injection

The method of inserting dependencies into an entity is known as injection. Injecting constructors is very popular. Dependency is injected as an argument to the constructor in this method.

Complete Example



## Output



# Google Guice - Method Injection

To add a value object as a dependency to an object, method injection is used.

## Output



**Google Guice Benefits**

Google Guice dependency injection is very simple to use and has a very short learning curve. Its simple advantages are that if you modify one dependency in a class, it affects all of the classes initialized with the dependency injection system without causing any of these classes to be modified.

# Google Guice - Field Injection

# Field injection is a technique for attaching a value object to a field of an object.

## Output



# Google Guice - Optional Injection

If a dependence occurs, optional injection implies injecting it. Method and field injections may be optionally based, and if they aren't, they should have a default value.



## Output

