Google Guice Dependency I Tutorial

APRIL 6, 2018 BY PANKAJ — 19 COMMENTS

Google Guice is the framework to automate the dependency injection in applications. If you have come across directly here, I would recommend you to check out Dependency Injection Example where we learned the problems with traditional approach of Object creation and implementation benefits of dependency injection.

In last tutorial, we learned how can we implement dependency injection in applications manually. But when number of classes grow in an application, it's better to look for some framework to automate this task.

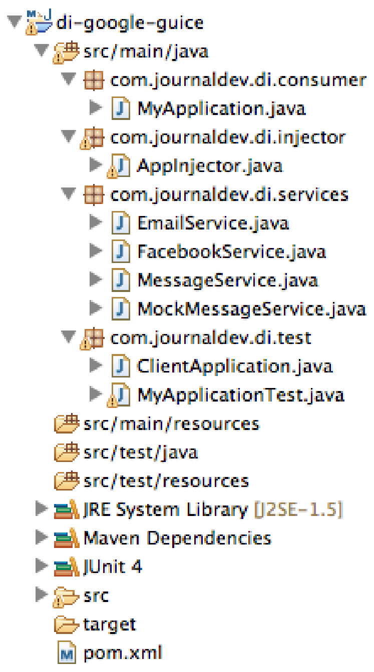
Google Guice is one of the leading frameworks whose main work is to provide automatic implementation of dependency injection. We will work on the same example from last post and learn how can we use Google Guice to automate the implementation process for dependency injection.

Google Guice dependencies are available on maven central, so for maven projects you can add below dependency for it.

```
<dependency>
        <groupId>com.google.inject</groupId>
        <artifactId>guice</artifactId>
        <version>3.0</version>
</dependency>
```

If you have a simple java application, then you can download the jar file from Google Guice Home Page on Google Code. Note that in this case you will also need to have it's transitive dependencies in the classpath or else you will get runtime exception.

For my example, I have a maven project whose project structure looks like below image.



```
di-google-guice
  src/main/java
    com.journaldev.di.consumer
      MyApplication.java
    com.journaldev.di.injector
      AppInjector.java
    com.journaldev.di.services
      EmailService.java
      FacebookService.java
      MessageService.java
      MockMessageService.java
    com.journaldev.di.test
      ClientApplication.java
      MyApplicationTest.java
  src/main/resources
  src/test/java
  src/test/resources
  JRE System Library [J2SE-1.5]
  Maven Dependencies
  JUnit 4
  src
  target
  pom.xml
```

Let's see each of the components one by one.

Table of Contents [hide]

Service Classes

```
package com.journaldev.di.services;

public interface MessageService {

	boolean sendMessage(String msg, String receipient);
}
```

`MessageService` interface provides the base contract for the services.

```
package com.journaldev.di.services;

import javax.inject.Singleton;

//import com.google.inject.Singleton;

@Singleton
public class EmailService implements MessageService {

	public boolean sendMessage(String msg, String receipient) {
		//some fancy code to send email
		System.out.println("Email Message sent to "+receipient+" with 
message="+msg);
		return true;
	}

}
```

`EmailService` is one of the implementation of `MessageService`. Notice that class is annotated with @Singleton annotation. Since service objects will be created through injector classes, this annotation is provided to let them know that the service classes should be singleton objects.

Google Guice 3.0 added the support for JSR-330 and we can use annotations from `com.google.inject` or `javax.inject` package.

Let's say we have another service implementation to send facebook messages.

```
package com.journaldev.di.services;

import javax.inject.Singleton;

//import com.google.inject.Singleton;

@Singleton
public class FacebookService implements MessageService {

        public boolean sendMessage(String msg, String receipient) {
                //some complex code to send Facebook message
                System.out.println("Message sent to Facebook user "+receipient+" with
message="+msg);
                return true;
        }

}
```

Consumer Class

Since we are implementing dependency injection in our application, we won't initialize the service class in application. Google Guice support both `setter-based` and `constructor-based` dependency injection. Our application class that consumes the service looks like below.

```
import com.journaldev.di.services.MessageService;

public class MyApplication {

        private MessageService service;

//      constructor based injector

//      @Inject
//      public MyApplication(MessageService svc){
//              this.service=svc;
//      }

        //setter method injector
        @Inject
        @Inject
        public void setService(MessageService svc){
                this.service=svc;
        }

        public boolean sendMessage(String msg, String rec){
                //some business logic here
                return service.sendMessage(msg, rec);
        }
}
```

Notice that I have commented the code for constructor based injection, this comes handy when your application provides some other features too that doesn't need service class object.

Also notice the @Injector annotation, this will be used by Google Guice to inject the service implementation class. If you are not familiar with annotations, check out **java annotations tutorial**.

Binding Service implementation

Obviously google guice will not know which service to use, we have to configure it by extending `AbstractModule` abstract class and provide implementation for `configure()` method.

```
package com.journaldev.di.injector;

import com.google.inject.AbstractModule;
import com.journaldev.di.services.EmailService;
import com.journaldev.di.services.FacebookService;
import com.journaldev.di.services.MessageService;

public class AppInjector extends AbstractModule {

        @Override
        protected void configure() {
                //bind the service to implementation class
                //bind(MessageService.class).to(EmailService.class);

                //bind MessageService to Facebook Message implementation
                bind(MessageService.class).to(FacebookService.class);

        }
}
```

As you can see that we can bind any of the implementation to service class. For example, if we want to change to EmailService we would just need to change the bindings.

Client Application

Our setup is ready, let's see how to use it with a simple java class.

```
package com.journaldev.di.test;

import com.google.inject.Guice;
import com.google.inject.Injector;

import com.journaldev.di.consumer.MyApplication;
import com.journaldev.di.injector.AppInjector;

public class ClientApplication {

        public static void main(String[] args) {
                Injector injector = Guice.createInjector(new AppInjector());

                MyApplication app = injector.getInstance(MyApplication.class);

                app.sendMessage("Hi Pankaj", "pankaj@abc.com");
        }

}
```

The implementation is very easy to understand. We need to create `Injector` object using Guice class createInjector() method where we pass our injector class implementation object. Then we use injector to initialize our consumer class. If we run above class, it will produce following output.

```
Message sent to Facebook user pankaj@abc.com with message=Hi Pankaj
```

If we change the bindings to `EmailService` in `AppInjector` class then it will produce following output.

```
Email Message sent to pankaj@abc.com with message=Hi Pankaj
```

JUnit Test Cases

Since we want to test MyApplication class, we are not required to create actual service implementation. We can have a simple Mock service implementation class like below.

```
package com.journaldev.di.services;

public class MockMessageService implements MessageService{

        public boolean sendMessage(String msg, String receipient) {
                return true;
        }

}
```

My JUnit 4 test class looks like below.

```
                        @Override
                        protected void configure() {


bind(MessageService.class).to(MockMessageService.class);
                        }
                });
        }

        @After
        public void tearDown() throws Exception {
                injector = null;
        }

        @Test
        public void test() {
                MyApplication appTest = injector.getInstance(MyApplication.class);
                Assert.assertEquals(true, appTest.sendMessage("Hi Pankaj",
"pankaj@abc.com"));;
        }

}
```

Notice that I am binding `MockMessageService` class to `MessageService` by having an anonymous class implementation of `AbstractModule`. This is done in `setUp()` method that runs before the test methods.

Download Google Guice Project

That's all for Google Guice Example Tutorial. Use of Google Guice for implementing dependency injection in application is very easy and it does it beautifully. It's used in Google APIs so we can assume that it's highly tested and reliable code. Download the project from above and play around with it to learn more.

« PREVIOUS
Java 8 Features with Examples

NEXT »
Serialization in Java – Java Serialization

About Pankaj
If you have come this far, it means that you liked what you are reading. Why not reach little more and connect with me directly on **Google Plus**, **Facebook** or **Twitter**. I would love to hear your thoughts and opinions on my articles directly.

Recently I started creating video tutorials too, so do check out my videos on **Youtube**.

FILED UNDER: JAVA

Comments

Binh Thanh Nguyen says

OCTOBER 20, 2017 AT 1:20 AM

Thanks, nice post

Reply

Srini says

APRIL 17, 2017 AT 5:09 AM

Very helpful.

Reply

Mayank Verma says

APRIL 10, 2017 AT 1:37 AM

Hi pankaj,

I imported the project in Intellij. Could you please tell me how to create a maven build conf. for that ?

Reply

sbeex says

MARCH 19, 2017 AT 1:41 PM

Thank you ! Guice for human arrived ! I used it for other projects but this time I found a really good introduction to understand the bases.

Reply

lawri says

AUGUST 16, 2016 AT 5:12 AM

My Main java class:

==================

package com.sample.test;

import com.google.inject.Guice;

import com.google.inject.Injector;

public class mymain {

public static void main(String[] args) {

// TODO Auto-generated method stub

Injector injector = Guice.createInjector(new AppInjectory());
ApplicationExample obj = injector.getInstance(ApplicationExample.class);
obj.sendMessage();
}
}
My interface
=============
package com.sample.test;
public interface MessageService {
boolean sendMessage(String msg, String receipient);
}
My config file
==================
package com.sample.test;
import com.google.inject.AbstractModule;
public class AppInjectory extends AbstractModule {
@Override
protected void configure() {
//bind the service to implementation class
//bind(MessageService.class).to(EmailService.class);
//bind MessageService to Facebook Message implementation
bind(MessageService.class).to(EmailService.class);
}
}
My appication file
===================
package com.sample.test;
import javax.inject.Inject;
public class ApplicationExample {
private MessageService service;
@Inject
public void setService(MessageService svc){
this.service=svc;
}
public void sendMessage() {
System.out.println("I am here");
service.sendMessage("welcome", "java");
}
}
My service class
=====================
package com.sample.test;
//import com.google.inject.Singleton;
import javax.inject.Singleton;

```
@Singleton
public class EmailService implements MessageService {
public boolean sendMessage(String msg, String receipient) {
//some fancy code to send email
System.out.println("Email Message sent to "+receipient+" with message="+msg);
return true;
}
}
```

Here I am getting NUll pointer exception .What wrong I did here.?please help to fix this issue.
ERROR:

```
Exception in thread "main" I am here
java.lang.NullPointerException
at com.sample.test.ApplicationExample.sendMessage(ApplicationExample.java:16)
at com.sample.test.mymain.main(mymain.java:13)
```

Reply

David Adkins says
MAY 25, 2016 AT 8:53 PM
Thank you Pankaj, This is my third tutorial of yours that I have gone through and I think they are very well written.
Reply

Pankaj says
JUNE 30, 2016 AT 10:20 AM
Thanks for the kind words David.
Reply

Chris Beach says
APRIL 7, 2016 AT 4:38 AM
Your "Shares" dialog is obscuring the text of this article
Reply

Pankaj says
JUNE 30, 2016 AT 10:21 AM
Thanks for the feedback, I have moved it to right side to fix it.
Reply

Rafal says
MARCH 21, 2016 AT 5:46 PM
You can use assertTrue(…) instead of assertEquals(true, …)in your test. Looks better
Reply

Pankaj says
JUNE 30, 2016 AT 10:22 AM
Yeah I can do that, thanks for the input. I will use it in my next JUnit examples.
Reply

Mark Muizer says
MARCH 15, 2016 AT 11:29 AM
property-based injection seams to work as well:
public class RestResource {
@Inject
private IemandsService service;
works like a charm
Reply

amol shinde says
NOVEMBER 8, 2015 AT 1:21 AM
thank you very much….explanat
ion is very good…
Reply

Igor says
AUGUST 5, 2015 AT 3:55 AM
Hello!
Can you please point discrepancies with this approach? I see one: @Singleton annotation.
Thank you!
Reply

Farhad says
MAY 12, 2015 AT 1:02 AM
So as per this tutorial, we can bind only one implementation at a time? What if I need to use FacebookMessageService AND EmailMessageService? How to handle that?
Reply

shashi says
MAY 26, 2015 AT 1:10 PM
Guice provides binding annotations for this purpose. Reference:
https://github.com/google/guice/wiki/BindingAnnotations
Reply

Ashish Dixit says
MAY 28, 2016 AT 7:39 AM
In this tutorial, MessageService is bound only to FacebookService which is practically useless because the interface is bound to just one implementation. If we want that the user has the flexibility to use any of the implementation, then we need to write multiple bind statements in configure() method and for making that possible, we need annotations like this :
bind(MessageService.class).annotatedWith("facebook").to(FacebookService.class);
bind(MessageService.class).annotatedWith("email").to(EmailService.class);
Now Guice knows that when to bind the MessageService interface to FacebookService and when with EmailService.
To call FacebookService implementation client side code will be like :
MyApplication app = injector.getInstance(@Named("facebook") MyApplication.class);
and to call EmailService ,
MyApplication app = injector.getInstance(@Named("email") MyApplication.class);
Hope this helps.
Reply

Sam says
SEPTEMBER 8, 2016 AT 3:06 PM
I tried using this solution but it does not work, I receive an error that says "Type annotations are illegal here" when attempting to creating one of the application instances.
Reply

Alex Wang says

FEBRUARY 23, 2015 AT 7:44 PM

good article about Guice

Reply

Leave a Reply

Your email address will not be published. Required fields are marked *

Comment

Name *

Email *

☐

Save my name, email, and website in this browser for the next time I comment.

POST COMMENT

Search for tutorials...

DOWNLOAD ANDROID APP

GET IT ON Google Play

CORE JAVA TUTORIAL

- **Sorting Objects in Java**
- **Understanding JDK, JRE and JVM**
- **Java Classloader Example Tutorial**
- **Java clone object**

RECOMMENDED TUTORIALS

Java Tutorials

- Java IO
- Java Regular Expressions
- Multithreading in Java
- Java Logging
- Java Annotations
- Java XML
- Collections in Java
- Java Generics
- Exception Handling in Java
- Java Reflection
- Java Design Patterns
- JDBC Tutorial

Java EE Tutorials

- Servlet JSP Tutorial
- Struts2 Tutorial
- Spring Tutorial
- Hibernate Tutorial
- Primefaces Tutorial
- Apache Axis 2
- JAX-RS
- Memcached Tutorial