CS309 Assignment of Dependency Injection

Authored by Hengcheng Zhu (zhuhc2016@mail.sustech.edu.cn)

Revised by CS309 Teaching Tram

Introduction

Dependency injection is commonly used in software development. Popular Java framework you use frequently are equipped with dependency injection, for example, Spring. To be an excellent stuent in CSE department of SUSTech, you should know not only how to use dependency injection frameworks but also how they work.

Suppose you have a class `A`, which depends on class `B`, `C`, and `D`, and class `C` depends on class `E` and `F`. When you need an instance of class `A`, in this case, you need to first create instances of class `E` and `F`, which will be used to create an instance of class `C`. After that, you should create instances of class `B` and `D`. Finally, you pass these instances to the constructor of `A`.

```java
public class A {
  public A(B serviceB, C serviceC, D serviceD) {
    // Creating an instance of class A requires instances of class B, C, and D.
  }
}
```

Creating instances of such classes is tedious and error-prone. The case would be much more complicated in real world applications. It is desired that these steps can be done automatically.

Dependency injection can act as a "smart factory" to help us achieve this goal. In this assignment, you will learn to build a tiny dependency injection framework by which you will have an impression of the internal mechamism of dependency injection.

Requirement

To get full score in this assignment, you are required to implement an interface provided by us.

```java
public interface Container {
    <T> void register(Class<T> serviceType);
    <T> void register(Class<T> serviceType, Class<? extends T> implementationType);
    <T> T resolve(Class<T> serviceType);
}
```

There are three methods in this interface. The first two are used for registering a service type, and, optionally, its corresponding implementation type. The last method is for getting an instance of the specified service type.

The following code snippet shows the example mentioned above, which demostrates the usage of the first and the last method in the `Container` interface.

```java
public class Example1 {
  public static void main(String[] args) {
    // Create an instance of container
    Container container = new ContainerImpl();

    // Register classes to the container
    container.register(A.class);
    container.register(B.class);
    container.register(C.class);
    container.register(D.class);
    container.register(E.class);
    container.register(F.class);

    // Get an object from container
    A aObject = container.resolve(A.class);
  }
}
```

The second method provides the ability to associate a service type, which is usually an abstract class or an interface, to a specific implementation type, which is usually a class that extends the abstract class or implemented the interface mentioned before. Note that when multiple implementations are associate to a service type, only the last one should take effect.

```java
public class Example2 {
  public static void main(String[] args) {
    // Create an instance of container
    Container container = new ContainerImpl();

    // Associate A with AImpl1
    container.register(A.class, AImpl1.class);

    // Get an object from container
    // Here aObject shoule be an instance of AImpl1
    A aObject = container.resolve(A.class);

    // Associate A with AImpl2
    container.register(A.class, AImpl2.class);

    // Get an object from container
    // Here aObject shoule be an instance of AImpl2
    A aObject = container.resolve(A.class);
  }
}
```

Dependencies can be specified by parameters in the constructor, or fields with `@Inject` annotation. Here is an example.

```java
public class A {

  private B bDep;

  @Inject
  private C cDep;

  public A(B bDep) {
    this.bDep = bDep;
  }
}
```

When dependencies can not be found, an `ServiceNotFoundException` should be thrown.

For simplicity, we guarantee that, in our test cases for grading:

- No cyclic dependency would occur

To be rigorius, you are required to carefully check the arguments before further processing. An `IllegalArgumentException` should be thrown if one of the following cases happen:

- `Null` argumnet value(s) for each method
- Register an interface or an abstract class for `serviceType` in the `register` method with one parameter, or for `implementationType` in the `register` method with two parameters.

Materials Provided

- `Container.java`: The interface to implement
- `ServiceNotFoundException.java`: The exception to be thrown when a service can not be found
- `Inject.java`: The annotation used to specify dependency

Submission

You should create a public class named `ContainerImpl` which implements the interface `Container`, and upload the file `ContainerImpl.java` to Sakai

You will **GET A ZERO** if one of the following happens:

- File name is not identical to the requirement (e.g., `containerImpl.java`, `ContainerImpl.Java`)
- Compilation fail
- Plagiarism