# Guice Integration Pitfall

The Wicket-Guice project is great and works as expected in most areas, so that anyone with a little experience with guice should have no problems using it within Wicket.
The main thing, Wicket-IOC does is create a Proxy to Guice-injected objects that makes it possible to serialize and deserialize (and relocate their delegates) without problems.

**Wicket 7.x** allows you to optionally use Objensis (instead of CGLib) by including it on the classpath to support **injecting concrete classes without a default constructor**. See **WICKET-5922** for details. (or the Github commit)

**For Wicket 6 and earlier:**

The one thing I stumbled upon, where Wicket does not behave like guice should be outlined here. If you´re not interested in the technical details, just take this advice:

When using Guice to inject objects that you reference directly (without an interface), and you´re using constructor injection on these Objects, add a **protected non-arg constructor** to make it work.

This is why:

Wicket-Guice, or - to be more precise -, Wicket-IOC uses a dynamically created Proxy to intercept the calls to the Guice-Injected reference. When dealing with an interface, the proxy is created with the JDK-provided proxy facility.
If - on the other hand - it is not, Wicket-IOC uses CGLib to create a subclass instead and adds an Interceptor, so that any calls to the newly created instance of this subclass can be rerouted to the original reference.
Within here is a litte problem: Wicket-IOC has to create an Instance of the newly created subclass.

Imagine the following code:

```
public class FooServiceImpl{
 @Inject public FooServiceImpl(BarService bar){
  [...]
 }
}
```

Guice happily creates an instance of FooServiceImpl where needed and injects a BarService if it is bound. You´ll now want to use it with Wicket Components:

```
public class XYPage extends WebPage
{
  @Inject FooServiceImpl foo;
}
```

What happens is, Wicket-IOC has to create a Subclass via CGLib and then create an Instance of that class. Even if the instance is never going to receive any call, is needed to make the interception work.
By the time of writing (Wicket-1.3rc1) it does this simply by assuming the class has a non-arg constructor. Even if Wicket-IOC would try to cheat by using the constructors that are visible instead, it would be hard to know, what values to use, if it did not mimic what guice does.
So if you really need that configuration (no interfaces, and no non-arg constructors) you could modify your class

```
public class FooServiceImpl{
 @Deprecated protected FooServiceImpl(){}
 @Inject public FooServiceImpl(BarService bar){
  [...]
 }
}
```

Guice will favor the parametrized Constructor, so that everything works as expected. One downside of this is the problem that arises from private final member that will now have to be assigned in order to get the class through the compiler.

Previous Issue for fixing this (marked as won't fix) was WICKET-1130.

Fixed in Wicket 7 in the following issue: WICKET-5922