**Dependency injection using Ninject**

"Ninject is a lightweight dependency injection framework for .NET applications. It helps you split your application into a collection of loosely-coupled, highly-cohesive pieces, and then glue them back together in a flexible manner. By using Ninject to support your software's architecture, your code will become easier to write, reuse, test, and modify."

## Important features of Ninject

1. Provide fluent interface
2. Light-weight (122 KB, version-3.0.015)
3. Faster because of lightweight code generation
4. Extensible (provide extensions for MVC, WCF and more)

## Modules and Kernels

## Modules

The *Modules* are the components that are used to register types. Modules act as independent segments of an application that takes care of binding all the interfaces with implementations of that segment. All the modules should implement the interface *INinjectModule*. Ninject comes with a built-in abstract class *NinjectModule* that implements this interface. This abstract class contains a single abstract method *Load* where we bind the interfaces with concrete types.

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7 | class WarriorModule : NijnectModule  {      public override Load()      {          Bind<IWeapon>().To<Sword>();      }  } |

## Kernel

In Ninject the *Kernel* is the component that controls everything. We can directly bind the interfaces with implementations into Kernel or we can pass them wrapped as modules. Whenever we need any implementation of the mapped interface we can get it right away from the Kernel. Ninject itself comes with a built-in Kernel called *StandardKernel*.

|  |  |
| --- | --- |
| 1  2  3 | // add bindings directly into the kernel  var kernel = new StandardKernel();  kernel.Bind<IWeapon>().To<Sword>(); |

or

|  |  |
| --- | --- |
| 1  2 | // supply bindings as a module  var kernel = new StandardKernel(new WarriorModule); |
| 1  2 | // get an IWeapon implementation  var weapon = kernel.Get<IWeapon>(); |



