Inversion of Control

This article is explaning about Depenedncy Injection or Inversion of Control. It is explaning with examples in C#.

- [Dhananjay Kumar]
- Jan 05 2009

What is the Problem?

Assumne there is a class called "Kids.cs". The purpose of this class is to maintain the name and age of the children of a specific person.

Kids.cs

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;

namespace IOC
{
   class Kids
   {
      private int age;
      private string name;
      public Kids(int age, string name)
      {
         this.age = age;
         this.name = name;
      }

      public override string ToString()
      {
         //return base.ToString();
         string m = this.age.ToString();
         return "KIDs Age " + m + " Kids Name " + name;
      }

   }
}
```

Now the requirement is that each person has Kids. So at the time of creation of a Person, the associated Kids should also be created. So this may be done by the following code.

Person.cs

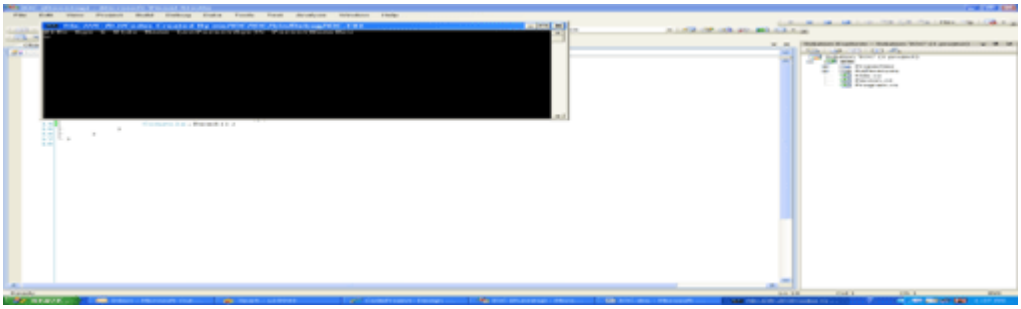
```
using System;
```

```
using System.Collections.Generic;
using System.Linq;
using System.Text;

namespace IOC
{
   class Person
   {
      private int age;
      private string name;
      private Kids obj;
      public Person(int personAge,string personName, int kidsAge, string kidsName)
      {
         obj = new Kids(kidsAge, kidsName);
         age = personAge;
         name = personName;
      }

      public override string ToString()
      {
         //return base.ToString();
         string s= age.ToString();
         Console.Write(obj);
         return "ParentAge" + s + " ParentName" + name;
      }
   }

}
```

Main.cs

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;

namespace IOC
{
   class Program
   {
      static void Main(string[] args)
      {
         Person p = new Person(35, "Dev", 6, "Len");
         Console.WriteLine(p);
         Console.Read();
      }
   }
}
```

Output



The code above is a typical example of a Composition design pattern or Object Dependency.

The following describes what Object Dependency is.

When an object needs another object then it is said that the object is dependent on the other object.

Let there are three classes Man, Woman and Children.

If Man -> Woman
And Woman -> Children
So Man -> Children

This is called Transitive Object Dependency.

Object Coupling and Object Dependency are the same term.

A good design should contain loose object coupling or object dependency.

Problems

The following are problems in the above approach:

1. It is using part of a relationship.
2. Here the Kids object has been created inside the constructor of the Person class. So here coupling is very high or tight.
3. If due to any reason the creation of the Kids object fails then the Person class also cannot be instantiated in its constructor itself.
4. If the Person object class is killed then by default the object of the Kids class is also killed.
5. If a new type of Kids has been created or a new property is being added in the Kids class then it needs code modification in the Person class also.
6. This also raises the chances of a Dangling Reference.
7. The Person class cannot work for the Default Constructor.
8. The Kid's class cannot work for the Default Constructor.

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;

namespace IOC
{
   class Person
   {
      private int age;
      private string name;
      private Kids obj;          (Refrence)
      public Person(int personAge,string personName, int kidsAge, string kidsName)
      {
         obj = new Kids(kidsAge, kidsName);        (known Concrete class)
         age = personAge;
         name = personName;
      }

      public override string ToString()
      {
         //return base.ToString();
         string s= age.ToString();
         Console.Write(obj);
         return "ParentAge" + s + " ParentName" + name;
```

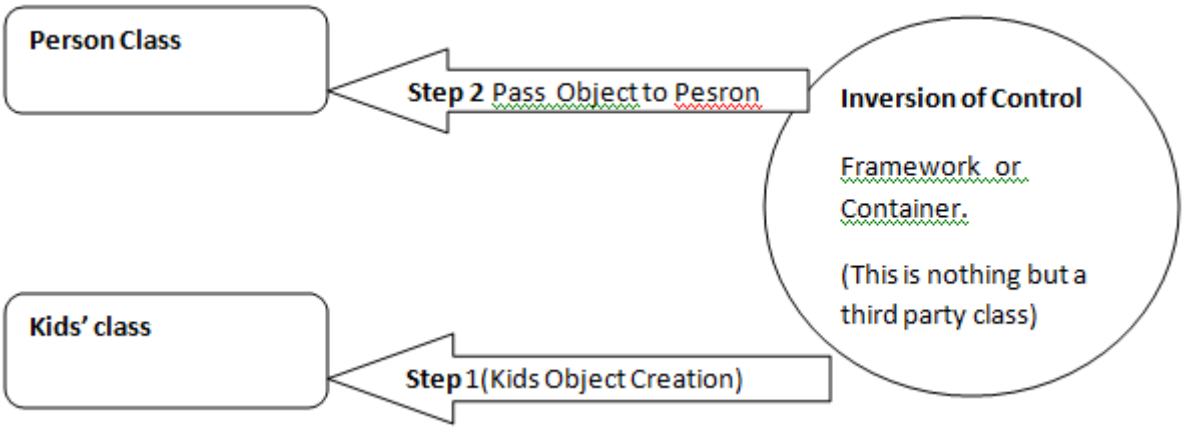
```
        }
    }

}
```

The preceding problem can be solved by the following:

1. Assign a task of object creation (Kids class) to some other entity, like another class or another function.
2. Assign a task of object creation to a third-party.

So, assigning the task of object creation to a third-party is somehow **inverting control** to the third-party. And this is called "Inversion of Control" (IOC).

In other words, the Inversion of Control Design Pattern could be defined as delegating the task of object creation to a third-party, to do low coupling among objects and to minimize dependency among objects.



So IOC says:

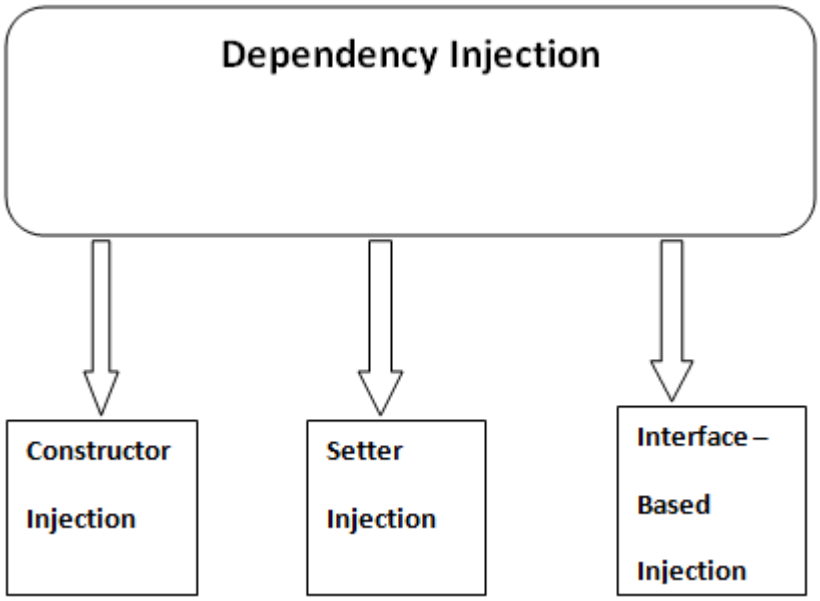
1. The main class (Pesron.cs) composes another class that should not directly depend on the implementation of the other class (Kids.cs).
2. There should be an abstraction among classes and the classes should fully depend upon that abstraction.
3. Abstraction could be either an interface or an abstract class.

Dependency Injection

Dependency Injection is the way to implement IOC as in the following:

1. Eliminates tight coupling among objects.
2. Makes the object and the application more flexible.
3. Facilitates creation of more loosely coupled objects and their dependencies.

"Dependency Injection isolates implementation of an object from the construction of the object on which it depends".



Constructor Injection

Here an object reference would be passed to the constructor of the business class Person. In this case, since the Person class depends on the Kids class. So the reference of the Kids class will be passed to the constructor of the Person class. So at the time of object creation of the Person class, the Kids class will be instantiated.

The following is the procedure to implement Constructor Injection.

Step 1: Create an interface

IBuisnessLogic.cs

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;

namespace IOC
{
   public interface IBuisnessLogic
   {

   }
}
```

1. This interface would be implemented by a Business Logic class. Here the Kids class is a business logic class.
2. Only the methods of the interface will be exposed to the Business Logic class (Person).

Step 2: Implement an interface to the Kids class.

Kids.cs

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;

namespace IOC
```

```
{
    public class Kids :IBuisnessLogic
   {
      private int age;
      private string name;
      public Kids(int age, string name)
      {
         this.age = age;
         this.name = name;
      }

      public override string ToString()
      {

         string m = this.age.ToString();
         return "KIDs Age " + m + " Kids Name " + name;
      }


   }
}
```

An object of the Kids class will be referenced by the Person class. So this class needs to implement the interface.

Step 3:

Make a reference of the interface in the Person class.

Person.cs

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;

namespace IOC
{
   class Person
   {
      private int age;
      private string name;
      IBuisnessLogic refKids;
      public Person(int personAge,string personName,IBuisnessLogic obj)
      {

         age = personAge;
         name = personName;
         refKids = obj;
      }

      public override string ToString()
      {
         string s= age.ToString();

         return "ParentAge" + s + " ParentName" + name;
      }
   }
}
```

Step 4:

Now to create a third-party class where the creation of the object will be done.

IOCClass.cs

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;

namespace IOC
{
   class IOCClass
   {
      IBuisnessLogic objKid = null;
      Person p;

      public void factoryMethod()
      {
         objKid = new Kids(12,"Ren");
         p= new Person (42,"David",objKid);
      }
      public override string ToString()
      {
         //return base.ToString();
         Console.WriteLine(p);
         Console.WriteLine(objKid);
         return "Displaying using Constructor Injection";

      }
   }
}
```

In the code above is a factorymethod() method where the object is getting created.

Step 5:

Now to use a third-party class at the client-side.

Program.cs

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;

namespace IOC
{
   class Program
   {
      static void Main(string[] args)
      {

         IOCClass obj = new IOCClass();
         obj.factoryMethod();
         Console.WriteLine(obj);


         Console.Read();
      }
   }

}
```

Disadvantage

1. In Constructor Injection, the business logic class cannot have a default constructor.
2. Once the class is instantiated, the object's dependency cannot be changed.

Setter Injection

This uses the Properties to inject the dependency. Here rather than creating a reference and assigning them in the constructor, it has been done in the Properties. By this way, the Person class could have a default constructor also.

Advantage:

1. It is more flexible than constructor injection.
2. Here the dependency of the object can be changed without creating an instance.
3. Here the dependency of the object can be changed without changing the constructors.
4. Setters have constructive and self-descriptive meaningful names that simplify the understanding and use of them.

Implementation:

Step 1: The same as Constructor Injection.

Step 2: The same as Constructor Injection.

Step 3: Pesron.cs

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;

namespace IOC
{
   class Person
   {
      private int age;
      private string name;
      //private Kids obj;
     private IBuisnessLogic refKids;
      public Person(int personAge,string personName)
      {

         age = personAge;
         name = personName;

      }
      public IBuisnessLogic REFKIDS
      {
         set
         {
            refKids = value;

         }
         get
         {
            return refKids;
         }
```

```
        }

        public override string ToString()
        {

            string s= age.ToString();

            return "ParentAge" + s + " ParentName" + name;
        }
    }

}
```

In the Person class is the property REFKIDS, that is setting and getting the value of the reference of the interface.

Step 4:

There are some changes in third-party class.

IOC.cs

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;

namespace IOC
{
    class IOCClass
    {
        IBuisnessLogic objKid = null;
        Person p;

        public void factoryMethod()
        {
            objKid = new Kids(12,"Ren");
            p= new Person (42,"David");
            p.REFKIDS = objKid;
        }
        public override string ToString()
        {
            //return base.ToString();
            Console.WriteLine(p);
            Console.WriteLine(objKid);
            return "Displaying using Setter Injection";

        }
    }

}
```

Step 5:

The same as constructor injection.