8 Lines of Code

Greg Young

Simplicity

I am too stupid to work otherwise.
Fancy code befuddles me.

```csharp
public class DeactivateInventoryItem
{
    private readonly ItemRepository repository;

    public DeactivateInventoryItem(ItemRepository repository)
    {
        this.repository = repository;
    }

    public void Deactivate(Guid id, string reason)
    {
        var item = repository.GetById(id);
        item.Deactivate(reason);
        repository.Save(item);
    }
}
```

```csharp
[Transactional]
[RequiresPermission("admin")]
[Logged]
[EatsExceptions]
[DoesBadThingsWhenYouArentWatching]
public class DeactivateInventoryItem
{
    private readonly ItemRepository repository;

    public DeactivateInventoryItem(ItemRepository repository)
    {
        this.repository = repository;
    }

    public void Deactivate(Guid id, string reason)
    {
        var item = repository.GetById(id);
        item.Deactivate(reason);
    }
}
```

Simplicity?

```xml
<bean id="moreComplexObject" class="example.ComplexObject">
<!-- results in a setAdminEmails(java.util.Properties) call -->
<property name="adminEmails">
  <props>
    <prop key="administrator">administrator@example.org</prop>
    <prop key="support">support@example.org</prop>
    <prop key="development">development@example.org</prop>
  </props>
</property>
<!-- results in a setSomeList(java.util.List) call -->
<property name="someList">
  <list>
    <value>a list element followed by a reference</value>
    <ref bean="myDataSource" />
  </list>
</property>
<!-- results in a setSomeMap(java.util.Map) call -->
<property name="someMap">
  <map>
    <entry key="an entry" value="just some string"/>
    <entry key ="a ref" value-ref="myDataSource"/>
  </map>
```

```csharp
[Transactional]
[RequiresPermission("admin")]
[Logged]
[EatsExceptions]
[DoesBadThingsWhenYouArentWatching]
public class DeactivateInventoryItem
{
    private readonly ItemRepository repository;

    public DeactivateInventoryItem(ItemRepository repository)
    {
        this.repository = repository;
    }


    public void Deactivate(Guid id, string reason)
    {
        var item = repository.GetById(id);
        item.Deactivate(reason);
    }
}
```

```csharp
[Transactional]
[Required("admin")]
[Logged]
[Ea...]
[DoesBad...entWatching]
public ...teInventoryItem
{
    private readonly ItemRepository repository;

    public DeactivateInventoryItem(ItemRepository repository)
    {
        this.repository = repository;
    }

    public void Deactivate(Guid id, string reason)
    {
        var item = repository.GetById(id);
        item.Deactivate(reason);
    }
}
```

```csharp
[Transactional]
[RequiresPermission("admin")]
[Logged]
[EatsExceptions]
[DoesBadThingsWhenYouArentWatching]
public class DeactivateInventoryItem
{
    private readonly ItemRepository repository;

    public DeactivateInventoryItem(ItemRepository repository)
    {
        this.repository = repository;
    }

    public virtual void Deactivate(Guid id, string reason)
    {
        var item = repository.GetById(id);
        item.Deactivate(reason);
    }
}
```

```
public object Foo()
{
    return this;
}
```

If you find you need an extension to your ide to understand what's going on. Its probably not simple.

What's the root of the problem?

```csharp
public void Deactivate(Guid id, string reason)
{
    var item = repository.GetById(id);
    item.Deactivate();
}


public void Reactivate(Guid id, DateTime effective,
                       string reason)
{
    var item = repository.GetById(id);
    item.Deactivate();
}


public void CheckIn(Guid id, int count)
{
    var item = repository.GetById(id);
    item.Deactivate();
}
```

No common interface!

```
public void Log(PointCut calledOn)
{
    logger.Log(calledOn.Name + ":" + calledOn.Parameters);
}
```

```csharp
public void Handle(DeactivateCommand c)
{
    var item = repository.GetById(c.id);
    item.Deactivate();
}


public void Handle(ReactivateCommand c)
{
    var item = repository.GetById(c.id);
    item.Reactivate();
}


public void Handle(CheckInCommand c)
{
    var item = repository.GetById(c.id);
    item.CheckIn(c.quantity);
}
```

```csharp
interface Handles<T> where T:Command
{
    void Handle(T command);
}
```

```csharp
class LoggingHandler<T>  : Handles<T> where T:Command
{
    private readonly Handles<T> next;

    public LoggingHandler(Handles<T> next)
    {
        this.next = next;
    }

    public void Handle(T command)
    {
        myLoggingFramework.Log(command);
        next.Handle(command);
    }
}


  var handler = new LoggingHandler<DeactivateCommand>(
                    new DeactivateCommandHandler(...)
                );
```

```csharp
public class DeactivateInventoryItem :Handles<DeactivateCommand>
{

    private readonly ItemRepository repository;

    public DeactivateInventoryItem(ItemRepository repository)
        {

            this.repository = repository;

        }


    public void Handle(DeactivateCommand command)
        {

            var item = repository.GetById(command.id);
            item.Deactivate(cmd.Reason);

        }
    }
```

```csharp
public class DeactivateInventoryItem :Handles<DeactivateCommand>
{
    private readonly ItemRepository repository;

    public DeactivateInventoryItem(ItemRepository repository)
    {
        this.repository = repository;
    }


    public void Handle(DeactivateCommand command)
    {
        var item = repository.GetById(command.id);
        item.Deactivate(cmd.Reason);
    }
}
```

```csharp
class Handlers
{
    public static void Handle(ItemRepository repository,
                              DeactivateCommand c)
    {
        var item = repository.GetById(c.id);
        item.Deactivate();
    }

    public static void Handle(ItemRepository repository,
                              ReactivateCommand c)
    {
        var item = repository.GetById(c.id);
        item.Deactivate();
    }

    public static void Handle(ItemRepository repository,
                              CheckInCommand c)
    {
        var item = repository.GetById(c.id);
        item.Deactivate();
    }
}
```

Back to the same problem!

```csharp
public static int Add(int a, int b)
{
    return a + b;
}
```

```csharp
public static int Add(int a, int b)
{
    return a + b;
}




            var add5 = x => Add(5, x);
```

```csharp
public static void Deactivate(ItemRepository repository,
                             DeactivateCommand c)
{

    var item = repository.GetById(c.id);
    item.Deactivate();
}
```

```csharp
public static void Deactivate(ItemRepository repository,
                             DeactivateCommand c)
{

    var item = repository.GetById(c.id);
    item.Deactivate();
}



    var nodepends = x => Deactivate(new ItemRepository(), x);
```

```csharp
void BootStrap()
{

    handlers.Add(x => Deactivate(new ItemRepository(), x));
    handlers.Add(x => Reactivate(new ItemRepository(), x));
    handlers.Add(x => CheckIn(new ItemRepository(),
                             new BarService(),
                             x));
}
```

```csharp
void BootStrap()
{
    handlers.Add(x => Deactivate(() => new ItemRepository(), x));
    handlers.Add(x => Reactivate(() => new ItemRepository(), x));
    handlers.Add(x => CheckIn(() => new ItemRepository(),
                             new BarService(),
                             x));
}
```

```
public static void Log<T>(T command, Action<T> next)
  where T:Command
  {

      myLoggingFramework.Log(command);
      next(command);

  }
```

```csharp
class LoggingHandler<T>  : Handles<T> where T:Command
{
    private readonly Handles<T> next;

    public LoggingHandler(Handles<T> next)
    {
        this.next = next;
    }

    public void Handle(T command)
    {
        myLoggingFramework.Log(command);
        next.Handle(command);
    }
}


  var handler = new LoggingHandler<DeactivateCommand>(
                  new DeactivateCommandHandler(...)
            );
```

```xml
<bean id="moreComplexObject" class="example.ComplexObject">
<!-- results in a setAdminEmails(java.util.Properties) call -->
<property name="adminEmails">
  <props>
    <prop key="administrator">administrator@example.org</prop>
    <prop key="support">support@example.org</prop>
    <prop key="development">development@example.org</prop>
  </props>
</property>
<!-- results in a setSomeList(java.util.List) call -->
<property name="someList">
  <list>
    <value>a list element followed by a reference</value>
    <ref bean="myDataSource" />
  </list>
</property>
<!-- results in a setSomeMap(java.util.Map) call -->
<property name="someMap">
  <map>
    <entry key="an entry" value="just some string"/>
    <entry key ="a ref" value-ref="myDataSource"/>
  </map>
```

```csharp
public static void Log<T>(T command, Action<T> next)
    where T:Command
    {

        myLoggingFramework.Log(command);
        next(command);

    }
```

Understand the problem a tool or idea solves well

If you need to add stuff to your IDE you are probably on the wrong path.

You own all code in your project.

Your boss doesn't care if the bug happened in someone else's library!