Design Defects & Restructuring

Week 5: 01 Oct 22

Rahim Hasnani

Agenda

- Dependency Injection
- Factory Method
- Introduction to Refactoring

MVC Continued – Dependency Injection

- Dependency Injection is a design pattern that implements IoC (Inversion of Control)
- Separates out the concerns of creating vs using the object, leading to loose coupling.
- Class A uses Class B => Class A depends on services of Class B
- Implementation alternatives:
  - Class A composes Class B and creates instance of class B
    - That's tightly coupled; Class A knows a LOT about class B
  - Create an interface/abstract-class to encapsulate class B services; Let class A create the concrete class and then user the interface
    - Somewhat better; still Class A is creating the concrete class

Dependency Injection Continued

Taken from https://munirhassan.com/2021/03/28/dependency-inversion-principle-and-the-dependency-injection-pattern/

```
public class Email
{
    public void SendEmail()
    {
        // code
    }
}
public class Notification
{
    private Email _email;
    public Notification()
    {
        _email = new Email();
    }
    public void PromotionalNotification()
    {
        _email.SendEmail();
    }
}
```

Dependency Injection Continued

```
public interface IMessageService
{
   void SendMessage();
}
public class Email : IMessageService
{
   public void SendMessage()
   {
      // code
   }
}
```

```
public class Notification
{
   private IMessageService
_iMessageService;

   public Notification()
   {
      _iMessageService = new Email();
   }
   public void PromotionalNotification()
   {
      _iMessageService.SendMessage();
   }
}
```

IDEAS?

Constructor Injection

```
public class Notification
{
   private IMessageService _iMessageService;

   public Notification(IMessageService _messageService)
   {
      this._iMessageService = _messageService;
   }
   public void PromotionalNotification()
   {
      _iMessageService.SendMessage();
   }
}
```

Property Injection

```
public class Notification
{
    public IMessageService MessageService
    {
        get;
        set;
    }
    public void PromotionalNotification()
    {

        if (MessageService == null)
        {
            // some error message
        }
        else
        {
            MessageService.SendMessage();

        }
    }
}
```

Method Injection

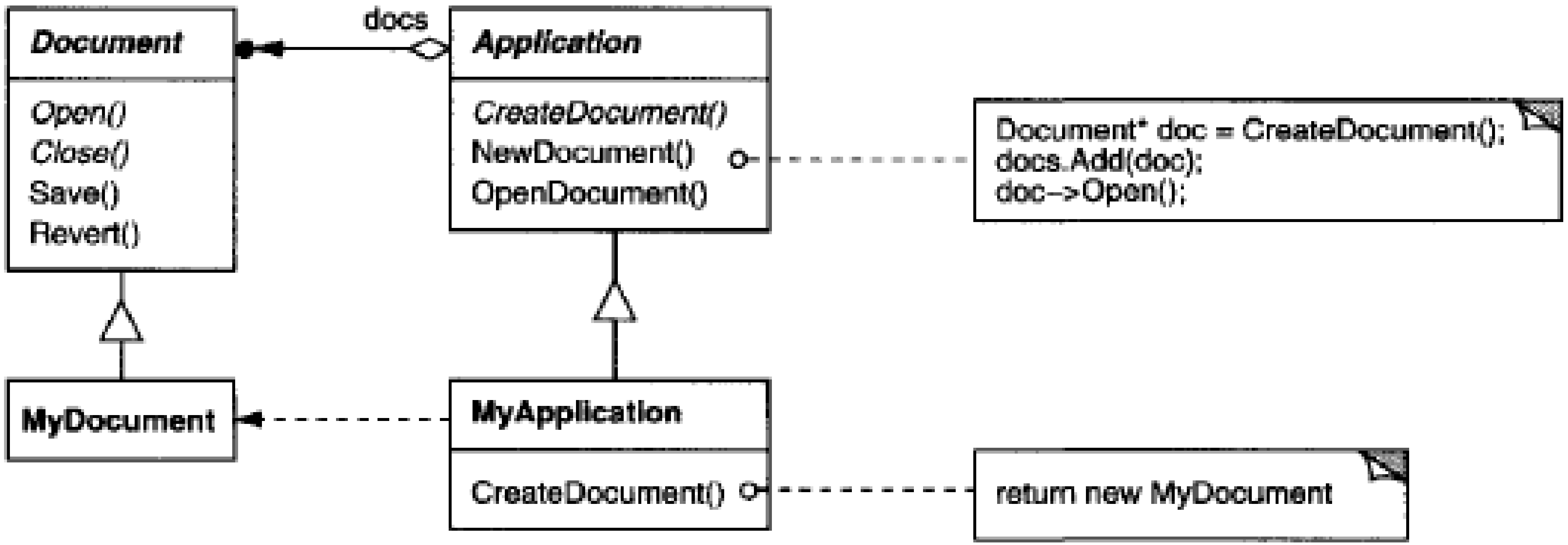
```
public class Email : IMessageService
{
    public void SendMessage()
    {
        // code for the mail send
    }
}
```

```
public class SMS : IMessageService
{
    public void SendMessage()
    {
        // code for the sms send
    }
}
```

```
public class Notification
{
    public void PromotionalNotification(IMessageService _messageService)
    {
        _messageService.SendMessage();
    }
}
```

Factory Method

- Intent
  - Define an interface for creating an object, but let subclasses decide which class toinstantiate. Factory Method lets a class defer instantiation to subclasses.
- Also Known As
  - Virtual Constructor
- Motivation
  - Consider a design framework where abstract classes Class A and Class D have a aggregation relation whereby Class A creates instances of Class D.
  - Since both classes A and D are abstract, they will be subclassed to provide implementation.
  - Since the knowledge about which subclass of D to instantiate is specific to subclass of A, at an abstract level A cannot predict which subclass of D to create.
  - The Factory method offers a solution here



```
Document* doc = CreateDocument();
docs.Add(doc);
doc->Open();
```

```
return new MyDocument
```