Refactoring Fundamentals

More Method Refactorings

Steve Smith
Ardalis.com
@ardalis

pluralsight
hardcore developer training

In This Course

- ~~What is Refactoring?~~
- ~~Why do it?~~
- ~~What's the process?~~
- ~~What are some tools that can assist with it?~~
- ~~What is a *Code Smell*?~~
- ~~*What are some examples of Code Smells?*~~
- What are some common refactorings?
- How does one apply them correctly?

Method-Related Refactorings (more)

- Preserve Whole Object
- Replace Parameter with Method
- Introduce Parameter Object
- Remove Setting Method
- Hide Method
- Replace Constructor with Factory Method
- Replace Error Code with Exception
- Replace Exception with Test
- Remove assignments to parameters
- Replace Method with Method Object
- Compose Method
- Substitute Algorithm

Preserve Whole Object

You are getting several values from an object and passing them as Parameters in a method call.

- **Create a new parameter for the whole object**
- ***Compile and Test***
- **Determine which parameters should come from the whole object**
- **Remove one parameter from the method**
  - Replace references to it in the method to calls on the whole object
- ***Compile and Test***
- **Repeat for each parameter than the whole object can provide**
- **Remove the code in the calling method that obtains the values before passing them into the method**
- ***Compile and Test***

Preserve Whole Object

```
int minCustomers = customerService.getMinCustomers();
int maxCustomers = customerService.getMaxCustomers();
string mostCommonRegion = customerService.getCommonRegion();
var plan = planFactory.Create(minCustomers, maxCustomers, region);
```

```
var plan = planFactory.Create(customerService);
```

Replace Parameter with Method

An object invokes a method, then passes the result as a parameter for a method. The receiver can also invoke this method.

- **Extract the calculation of the parameter into a method**
  - If one does not already exist
- **Replace references to the parameter with references to this method**
- **Apply Remove Parameter on the replaced parameter**

If a method can get a value that is passed in as a parameter by another means, it should.

Martin Fowler

Replace Parameter with Method

```
int basePrice = _quantity * _itemPrice;
decimal discountLevel = getDiscountLevel();
decimal finalPrice = getDiscountedPrice(basePrice, discountLevel);
```

```
int basePrice = _quantity * _itemPrice;
decimal finalPrice = getDiscountedPrice(basePrice);
```

Introduce Parameter Object

You have a group of parameters that naturally go together.

- **Create a new class to represent the parameter group**
  - Make the class immutable
- ***Compile***
- **Use Add Parameter to add the new class to the method's list of parameters**
  - Pass in null for this parameter from all callers for now
- **Remove one of the parameters that is now in the new object**
  - Modify the callers and method body to use the parameter object
- ***Compile and Test. Repeat for each parameter.***
- **Consider moving behavior into the new object (*Move Method*)**

Introduce Parameter Object

```
// fetch address variables from UI controls
var cart = new Cart();
cart.Add(items);

cart.CheckOut(customerEmail, customerName, customerStreet1, customerStreet2,
    customerCity, customerState, customerZIP, customerCountry,
    customerCreditCardNumber, customerCreditCardExpirationMonth,
    customerCreditCardExpirationYear, customerCreditCardSecretNumber);



var cart = new Cart();
cart.Add(items);
var address = new Address(customerStreet1, customerStreet2, customerCity,
    customerState, customerZIP, customerCountry);
var creditCard = new CreditCardInfo(customerCreditCardNumber,
    customerCreditCardExpirationMonth, customerCreditCardExpirationYear,
    customerCreditCardSecretNumber);

cart.CheckOut(customerEmail, customerName, address, creditCard);
```

Remove Setting Method

A property should be set at creation time and never altered.

- **Change the protection level of the set method to private**
- ***Compile and Test***
  - If any child classes are trying to set this property, consider changing its protection level to protected

Remove Setting Method

```
public class Account
{
    public int Id { get; set; }
}


public class Account
{
    public int Id { get; private set; }
    public Account(int id)
    {
        this.Id = id;
    }
}
public class PremiumAccount : Account
{
    public PremiumAccount(int id) : base(id)
    {
    }
}
```

Hide Method

A method is not used by any other class.

- Check regularly for opportunities to make a method more private
- Make a method (or property accessor) private (or protected or internal)
- *Compile*

Replace Constructor with Factory Method

You want to do more than simple construction when you create an object.

- **Create a new factory method**
  - Its signature should match the constructor it is replacing
- **Replace all calls to the constructor with calls to the factory method**
- ***Compile and Test after each replacement***
- **Declare the constructor private**
- ***Compile***

Replace Constructor with Factory Method

```
public enum EmployeeType
{
    Engineer,
    Salesman,
    Manager
}

public class Employee
{
    private readonly EmployeeType _employeeType;

    public Employee(EmployeeType employeeType)
    {
        this._employeeType = employeeType;
    }
}
```

Replace Constructor with Factory Method

```
public enum EmployeeType
{
    Engineer,
    Salesman,
    Manager
}
public class Employee
{
    private readonly EmployeeType _employeeType;

    private Employee(EmployeeType employeeType)
    {
        this._employeeType = employeeType;
    }

    public static Employee Create(EmployeeType employeeType)
    {
        return new Employee(employeeType);
    }
}
```

Replace Constructor with Factory Method

```
public abstract class Employee
   {
       public static Employee Create(EmployeeType employeeType)
       {
           switch (employeeType)
           {
               case EmployeeType.Engineer: return new Engineer();
               case EmployeeType.Salesman: return new Salesman();
               case EmployeeType.Manager:  return new Manager();
               default:
                   break;
           }
           throw new ArgumentOutOfRangeException("employeeType");
       }
   }

   public class Engineer : Employee { }
   public class Salesman : Employee { }
   public class Manager : Employee { }
```

Replace Error Code with Exception

A method returns a special code to indicate an error.

- **Identify the error code being returned in the method**
- **Replace it with throwing a meaningful exception**
- ***Compile***
- **Modify error handling in the client code to handle the exception in a catch {} block**
  - If this is too large a change due to number of clients:
    - Create a new method that throws the exception
    - Change the original method to call this one, and return the error code from its catch{} block
    - Gradually change client code to use the new method
- ***Compile and Test***

Replace Error Code with Exception

```
public class Account
{
    private decimal _currentBalance;

    public int Withdraw(decimal amount)
    {
        if (amount > _currentBalance)
        {
            return -1;
        }
        _currentBalance -= amount;
        return 0;
    }
}
```

Replace Error Code with Exception

```
public class Account
   {
       private decimal _currentBalance;

       public void Withdraw(decimal amount)
       {
           if (amount > _currentBalance)
           {
               throw new InsufficientFundsException("Insufficient funds");
           }
           _currentBalance -= amount;
       }
   }

   public class InsufficientFundsException : ApplicationException
   {
       public InsufficientFundsException(string message) : base(message)
       {}
   }
```

Replace Error Code with Exception

```
public int Withdraw(decimal amount)
{
    try
    {
        Withdraw2(amount);
    }
    catch (InsufficientFundsException ex)
    {
        return -1;
    }
    return 0;
}
```

Remove Assignments to Parameters

The code assigns to a parameter.

- **Identify the assignment to the parameter**
- **Declare a temp of the appropriate type just before the assignment**
  - Initialize the temp's value to the value of the parameter
- **Change the assignment to the parameter to the temp**
  - Replace all future instances of the parameter with the temp
- ***Compile and Test***

Replace Exception with Test

A method is throwing an exception on a condition the caller could have checked first (and which is not exceptional).

- **Write an if() statement to test that the exception state will occur**
  - Put the "sad path" code (from the catch statement) in the if block
  - Put the original try-catch code in the else{} condition
- **The catch code should now never be called**
  - Write tests to confirm, if necessary
- ***Compile and Test***
- **Remove the try-catch**
  - Leave only the body of the try{} in the else{}
- ***Compile and Test***
- **If desired, reverse the *if* and *else* blocks (Reverse Conditional)**

Replace Exception with Test

```
public string GetGreeting()
{
    Customer customer = null;
    try
    {
        customer = Cache[_customerKey];
        return customer.Greeting();
    }
    catch (NullReferenceException)
    {
        customer = new Customer(_customerKey);
        Cache.Add(_customerKey, customer);
        return customer.Greeting();
    }
}
```

Replace Exception with Test

```
public string GetGreeting()
{
    Customer customer = Cache[_customerKey];
    if (customer == null)
    {
        customer = new Customer(_customerKey);
        Cache.Add(_customerKey, customer);
    }
    return customer.Greeting();
}
```

Replace Method with Method Object

You have a long method that uses local variables in such a way that you cannot apply Extract Method.

- **Create a new class, name it after the method**
- **Create a new readonly field for the object that hosted the original method (perhaps *source*)**
  - Create additional fields for all parameters
- **Give the new class a constructor that takes the source object and each parameter**
- **Give the new class a new method, Compute()**
- **Copy the body of the original method into Compute()**
  - Use the source object field for any methods called on the original object
- ***Compile***
- **Replace the old method body**
  - Create the new object
  - Call Compute()
- ***Compile and Test***

Replace Method with Method Object

Demo

Compose Method

You can't rapidly understand a method's logic.

Guidelines

- *Think small*
- *Remove duplication and dead code*
- *Communicate intent*
- *Simplify*
- *Use the same level of abstraction*

Compose Method

```
public void Add(object item)
{
    if (!this.ReadOnly)
    {
        int newSize = this._size + 1;
        if (newSize > elements.Length)
        {
            object[] newElements = new object[elements.Length + 10];
            for (int i = 0; i < _size; i++)
            {
                newElements[i] = elements[i];
            }
            elements = newElements;
        }
        elements[_size++] = item;
    }
}
```

Compose Method

```
public void Add(object item)
{
    if (this.ReadOnly)
    {
        return;
    }
    if (AtCapacity())
    {
        Grow();
    }
    AddItem(item);
}

private bool AtCapacity()
{
    return _size + 1 > elements.Length;
}
```

Substitute Algorithm

You want to replace an algorithm with one that is clearer.

- **Prepare an alternative algorithm**
  - Probably in a separate method
- ***Compile and Test* the new algorithm to ensure it is correct**
- **Replace the original algorithm with the new one**
- ***Compile and Test***

Substitute Algorithm

```
public string Generate(int input)
{
    if(input < 2)
    {return "1";}
    if (input == 3)
    {return "Fizz";}
    return "2";
}
```

Substitute Algorithm

```
public string Generate(int input)
{
    if (input == 3)
    {
        return "Fizz";
    }
    return input.ToString();
}
```

Summary

- Preserve Whole Object
- Replace Parameter with Method
- Introduce Parameter Object
- Remove Setting Method
- Hide Method
- Replace Constructor with Factory Method
- Replace Error Code with Exception
- Replace Exception with Test
- Remove assignments to parameters
- Replace Method with Method Object
- Compose Method
- Substitute Algorithm

References

Books

Refactoring http://amzn.to/110tscA

Refactoring to Patterns http://amzn.to/Vq5Rj2

Web

Refactoring Catalog http://www.refactoring.com/catalog/

Thanks!

Steve Smith

Ardalis.com

Twitter: @ardalis

To Teach Is To Learn Twice

pluralsight
hardcore developer training