Introduction to Refactoring

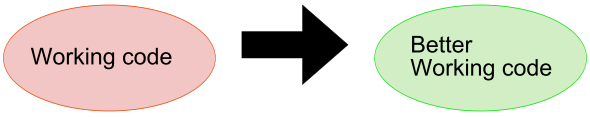
- What is Refactoring?
- Refactoring Techniques

Copyright © Descriptor Systems, 2001-2014. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Copyright © Descriptor Systems, 2001-2014. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

What is Refactoring?

- Refactoring is the act of improving existing code without changing its behavior
- Refactoring makes code easier to understand and maintain





The shirt is available from:

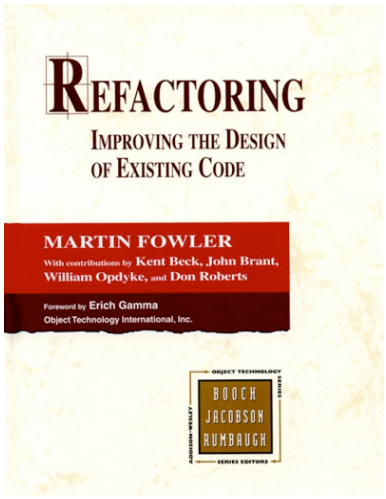
http://t-shirts.cafepress.com/item/born-to-refactor-jersey/26911153

(We have no association with this site.)

Copyright © Descriptor Systems, 2001-2014. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

History of Refactoring

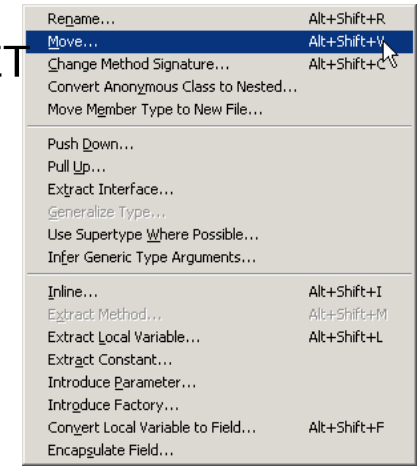
- Refactoring has been practiced since the dawn of programming, but only fairly recently has been formalized
- The most recognized book on the subject is **Refactoring: Improving the Design of Existing Code** by Martin Fowler



Copyright © Descriptor Systems, 2001-2014. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

IDE Support for Refactoring

- Many modern integrated development environments provide automated support for many common refactorings
  - Eclipse and IDEs based on it (IBM RAD, etc.)
  - IntelliJ IDEA
  - NetBeans
  - Visual Studio.NET
- Most IDEs let you preview the changes before applying them



The screen shot shows a drop-down menu from the Eclipse IDE, demonstrating some of the automated refactorings it supports.

Copyright © Descriptor Systems, 2001-2014. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

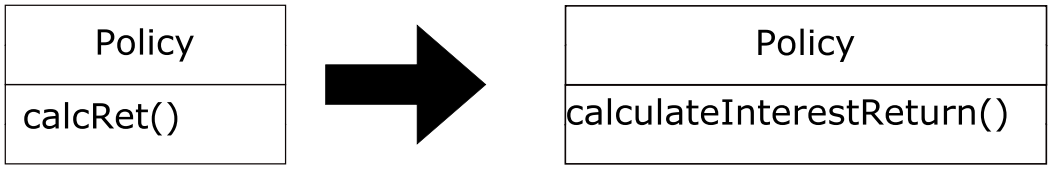
Refactoring Techniques

- Fowler identified many common techinques in his book
- The techniques generally fall into three categories:
    - Techniques that increase abstraction
    - Techniques that improve the names or location of code
    - Techniques for breaking apart code into more manageable units
- There is a complete list of refactorings at www.refactoring.com/catalog/index.html

Copyright © Descriptor Systems, 2001-2014. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Rename Method or Field

- This refactoring improves code by letting you ensure that a method's name matches its purpose
- Most IDEs provide this refactoring and replace method invocations with the new name



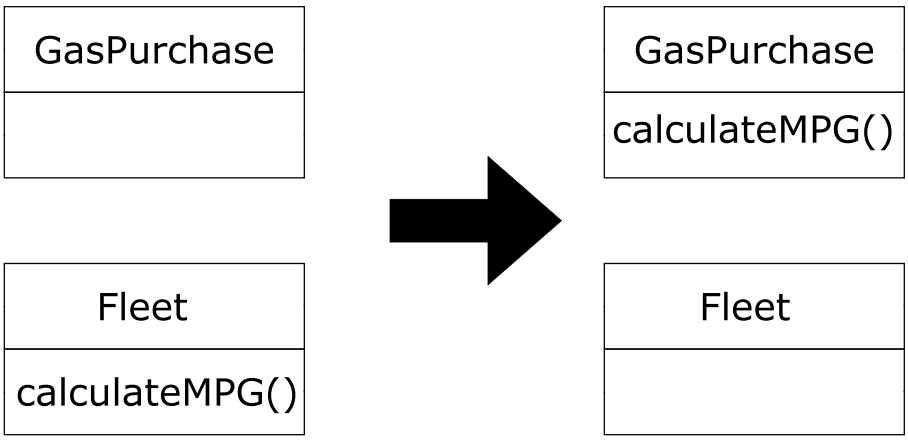
This is perhaps the simplest refactoring, but still useful.

Note that having an automated tool for this refactoring is quite handy since it frees you from manually needing to scan other code to find calls to the renamed method.

Copyright © Descriptor Systems, 2001-2014. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Move Method or Field

- This refactoring can improve the **cohesiveness** of classes by moving a method into the class where it logically belongs
- IDEs often have restrictions on this automatic refactoring



2 - 7

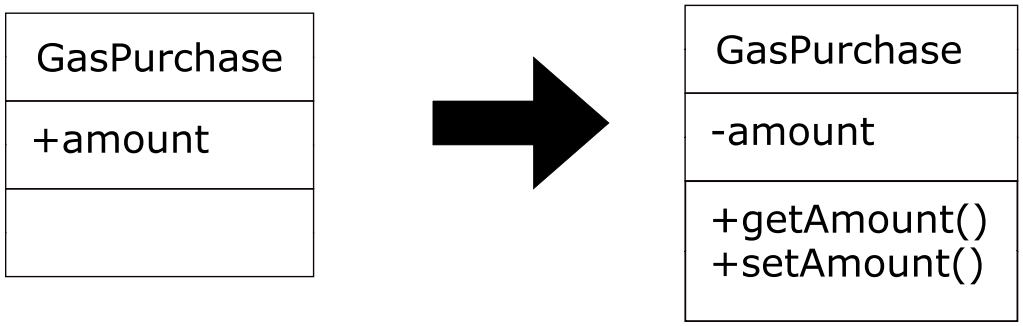
Cohesiveness is a measurement that determines how well a class holds together. The basic thought is that a class "should do only one thing and do it well".

There is another design metric known as GRASP patterns that state that you should place a behavior in the class that has the state to support the behavior. If you find that you've violated this principle, move method can help you clean it up.

Copyright © Descriptor Systems, 2001-2014. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Encapsulate Field

- This refactoring improves the "information hiding" of a class by making public fields private and providing "get/set" methods



While there is some controversy over the benefits of "get/set" methods, some situations require them. For example, a JavaBean only exposes properties via get/set methods.

Copyright © Descriptor Systems, 2001-2014. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Extract Method

- This refactoring lets you change inline code into a method whose name describes the code
- It also makes code re-use easier
- Many IDEs can scan to find occurences of the extracted code and replace them with method invocations

```
void printPurchases(List<GasPurchase> gp)
{
  System.out.println("Gas Purchase Info");
  System.out.println(" as of " + new java.util.Date());

  double total = 0;
  int count = 0;
  for (GasPurchase g : gp)
  {
    total += g.getAmount();
    count++;
  }

  System.out.println("Average: " + total/count);
}
```



```
void printPurchases(List<GasPurchase> gp)
{
  printBanner();
  printAverage(gp)
}

void printBanner()
{
  System.out.println("Gas Purchase List");
  System.out.println(" as of " + new java.util.Date());
}

void printAverage(List<GasPurchase> gp)
{
  double total = 0;
  int count = 0;
  for (GasPurchase g : gp)
  {
    total += g.getAmount();
    count++;
  }
  System.out.println("Average: " + total/count);
}
```

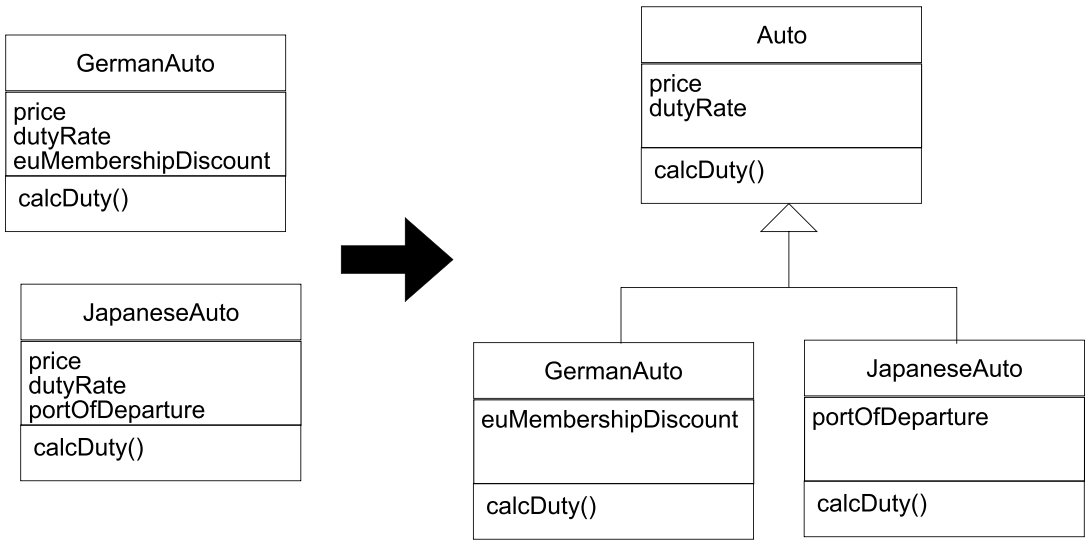
This refactoring lets you create methods that are easier to understand, maintain and override in subclasses. According to Fowler, this is one of the most commonly used refactorings.

There is an "opposite" refactoring named Inline Method, which is useful if you end up with tiny methods whose body by itself is just as clear as the method name.

Copyright © Descriptor Systems, 2001-2014. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Extract Superclass

- This refactoring moves commonalities from subclasses to a new superclass
- This removes duplication and makes class hierarchies easier to maintain



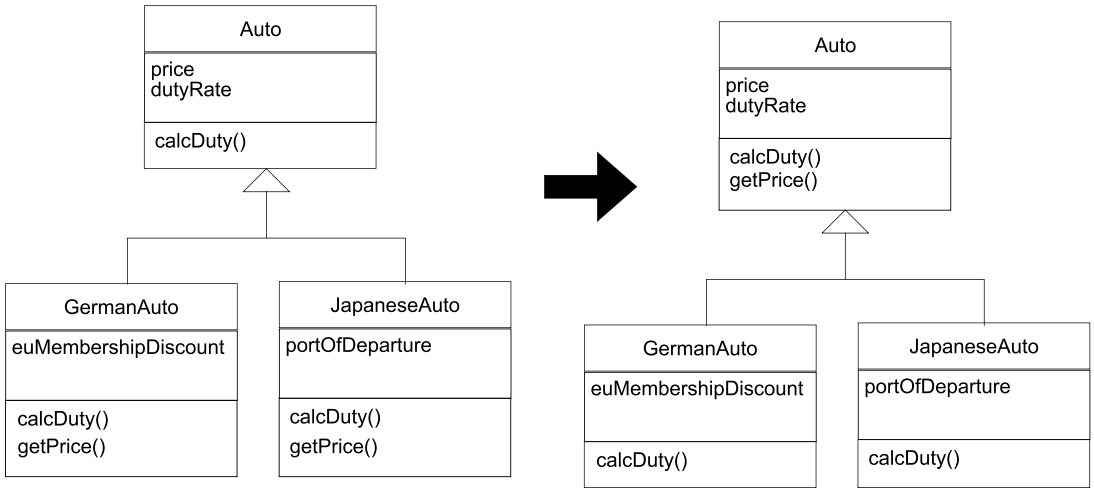
This refactoring also introduces the possibility of polymorphism.

As an alternative to Extract Superclass, consider Extract Class which uses delegation rather than inheritance to eliminate duplication.

Copyright © Descriptor Systems, 2001-2014. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Move Method to Superclass (Pull Up)

- This refactoring moves a method from existing subclasses to an existing superclass
- There is a similar refactoring named **Pull Up Field**



While this refactoring is sometimes very obvious, in many cases you will have duplication in multiple methods that are not identical. In that case, you can apply this refactoring after modfying the method(s) as necessary so that they are identical.

Copyright © Descriptor Systems, 2001-2014. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Chapter Summary

In this chapter, you learned:

- The basics of refactoring
- Common refactoring techniques

Copyright © Descriptor Systems, 2001-2014. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Copyright © Descriptor Systems, 2001-2014. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum