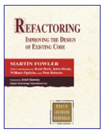
Advanced Object-Oriented Design
Lecture 6

Software Refactoring

Primitive transformations

Bartosz Walter
<Bartek.Walter@man.poznan.pl>

Refactoring Template by Fowler



Name

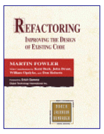
Summary

Goal

Mechanics

Examples

Refactoring Template by Fowler



Name
Extract Method

Summary
Extracting a new method from selected code snippet

Goal
Create a common method for further reuse

Mechanics
- *create a new method and copy the piece of code to it*
- *look for modified local variables*
- *pass the local variables to the new method*
- *replace the old code by the new method call*
- *compile and test*

Example
...

Catalog of software refactorings

Refactorings related to
- long parameter list
- feature envy
- data clumps

Add Parameter

Name
Add Parameter

Summary
A method needs more information from the caller

Goal
Add an parameter to the method signature

Mechanics
- check if method is not polymorphic
- declare a new method with the added parameter
- copy the old body over to the new one
- compile
- change the old method so that it delegates to the new one
- compile & test
- replace references to the old method with the new one
- remove old method (if not a part of an interface)
- compile & test

Example: Add Parameter

Example
```
void methodA(int param1, List param2) {
   //... body
}
```

```
void methodA(Map param0, int param1, List param2) {
   //...body
}
void methodA(int param1, List param2) {
   methodA(null, param1, param2);
}
```

Remove Parameter

Name
Remove Parameter

Summary
A parameter is no longer used in method body

Goal
Remove the parameter from the method signature

Mechanics
- check if method is not polymorphic
- declare a new method without the parameter to be removed
- copy the old body over to the new one
- compile
- change the old method so that it delegates to the new one
- compile & test
- replace references to the old method with the new one
- remove old method (if not a part of an interface)
- compile & test

Example: Remove Parameter

Example
```
void methodA(Map param0, int param1, List param2) {
   //... body
}
```

```
void methodA(Map param0, int param1, List param2) {
   methodA(param1, param2);
}
void methodA(int param1, List param2) {
   //...body
}
```

Preserve Whole Object

Name
Preserve Whole Object

Summary
Several values from an object are passed as parameters in a method call

Goal
Pass the reference to the whole object instead

Mechanics
- create a new parameter in the method for a whole object
- compile & test
- replace references to one of params by invoking an appropriate method on the whole object parameter
- delete the parameter, compile & test

Example: Preserve Whole Object

Example
```
public void printPersonalCard(String surname, String name, String phone) {
   System.out.println("Surname: " + surname);
   System.out.println("Name: " + name);
   System.out.println("Phone: " + phone);
}

printPersonalCard(p.getSurname(), p.getName(), p.getPhone());
```

Example: Preserve Whole Object

```
public void printPersonalCard(Person person, String surname, String name,
   String phone) {
   System.out.println("Surname: " + person.getSurname());
   System.out.println("Name: " + person.getName());
   System.out.println("Phone: " + person.getPhone());
}

printPersonalCard(person, null, null, null);
```

```
public void printPersonalCard(Person person) {
   // System.out.println...
}

printPersonalCard(person);
```

Replace Parameter with Explicit Methods

Name
Replace Parameter with Explicit Methods

Summary
A method runs different code depending on the values of an enumerated parameter

Goal
Create a separate method for each value of the parameter

Mechanics
- create an explicit method for each value of the parameter
- for each leg of conditional call the appropriate new method
- compile & test
- make each caller of the old method to call appropriate new method
- remove old method

Example: Replace Parameter with Explicit Methods

Example

```
public void draw(FigureType type, Size size) {
    if (type == FigureType.SQUARE) {
        // draw square
    } else if (type == FigureType.TRIANGLE) {
        // draw triangle
    } else if (type == FigureType.CIRCLE) {
        // draw circle
    }
}
```

Example: Replace Parameter with Explicit Methods

```
public void drawSquare(Size size) {
      // draw square
}
public void drawTriangle(Size size) {
      // draw triangle
}
public void drawCircle(Size size) {
      // draw circle
}
```

Replace Parameter with Method

Name
Replace Parameter with Method

Summary
An object invokes a method, passing the result as a parameter for another method

Goal
Remove the parameter and let the receiver call the method

Mechanics
- if necessary, extract the calculation of the parameter into a method
- replace reference to the parameter in method bodies with calls to the newly created method
- compile & test
- remove unreferenced parameter

Example: Replace Parameter with Method

Example

```
int basePrice = quantity * itemPrice;
double discountLevel = getDiscountLevel();
double finalPrice = discountedPrice(basePrice, discountLevel);
```

```
int basePrice = quantity * itemPrice;
double finalPrice = discountedPrice(basePrice);

public double discountedPrice(int basePrice) {
   double discountLevel = getDiscountLevel();
   //....
}
```

Catalog of software refactorings

Related code smells
- temporary field
- long method

Inline Temp

Name
Inline Temp

Summary
A temp is assigned only once with a simple expression

Goal
Replace all references to that temp with the expression

Mechanics
- declare the temp as final and compile
- replace all the reference to the temp with right-hand side of the assignm.
- compile & test after each change
- remove the declaration and the assignment of the temp
- compile & test

Example: Inline Temp

Example 1

```
final double basePrice = order.basePrice();
return basePrice > 1000;
```

```
return (order.basePrice() > 1000);
```

Example 2

```
StringTokenizer st = new StringTokenizer("ala ma kota", " ");
String token = st.next();
System.out.println("Token = " + token);
System.out.println("Token = " + token);
```

```
System.out.println("Token = " + st.next());
System.out.println("Token = " + st.next());
```

Separate Query from Modifier

Name
Separate Query From Modifier

Summary
Method both returns a value and changes the state of an object

Goal
Create two methods, one for query and another for modifier

Mechanics
- create a query that returns the same value as the original method
- modify the original method so that it returns the result of the query
- compile & test
- replace calls to the original method with calls to the query
- add a call to the original method before every query call
- make the original method have a *void* return type, remove return expressions

Introduce Explaining Variable

Name
Introduce Explaining Variable

Summary
There is a very complicated expression

Goal
Store result of the expression in a temp variable with a meaningful name

Mechanics
- declare final temp and assign the expression to it
- replace expression with the temp
- compile & test

Example: Introduce Explaining Variable

Example

```
if (os.toUpperCase().indexOf("Windows") > -1) &&
   browser.toUpperCase().indexOf("Gecko") > -1 &&
   wasInitialized() && resize > 0) {
      //....
}
```

```
final boolean isWindows = os.toUpperCase().indexOf("Windows") > -1;
final boolean isGecko = browser.toUpperCase().indexOf("Gecko") > -1
final boolean wasResized = resize > 0;

if (isWindows && isGecko && wasInitialized() && wasResized) {
   // ....
}
```

Replace Temp with Query

Name
Replace Temp with Query

Summary
A temporary variable holds a result of an expression

Goal
Extract the expression into a method

Mechanics
- declare the temp as final (non-modifiable)
- compile
- extract the right-hand expression into a method
- compile & test
- apply *Inline Temp* refactoring

Example: Replace Temp with Query

Example

```
final boolean isWindows = os.toUpperCase().indexOf("Windows") > -1;
final boolean isGecko = browser.toUpperCase().indexOf("Gecko") > -1
final boolean wasResized = resize > 0;

if (isWindows && isGecko && wasInitialized() && wasResized) {
   // handle
}
```

Example: Replace Temp with Query

```
private boolean isWindows() {
   return os.toUpperCase().indexOf("Windows") > -1;
}

private boolean isGecko() {
   return browser.toUpperCase().indexOf("Gecko") > -1;
}

private boolean wasResized() {
   return resize > 0;
}

if (isWindows() && isGecko() && wasInitialized() && wasResized()) {
   // handle
}
```

Reduce Scope of Variable

Name
Reduce Scope of Variable

Summary
There is a local variable declared in a larger scope than it is necessary

Goal
Reduce the scope of the variable so that it is only visible in the scope where it is used

Mechanics
- move the declaration of the variable to the scope where that variable is used
- compile and test

Split Temporary Variable

Name
Split Temporary Variable

Summary
A temporary variable is assigned more than once

Goal
Provide a separate variable for each assignment

Mechanics
- change name of the temp at declaration
- declare new final temp variable
- change all references of the temp up to its second (third,...) assignment
- compile & test
- repeat for further assignments

Example: Split Temporary Variable

```
double getDistanceTravelled(int time) {
  double result = 0;
  double acc = primaryForce / mass;
  int primaryTime = Math.min(time, delay);
  result = 0.5 * acc * primaryTime * primaryTime;
  int secondaryTime = time – delay;
  if (secondaryTime > 0) {
    double primaryVelocity = acc * delay;
    acc = (primaryForce + secondaryForce) / mass;
    result += primaryVelocity * secondaryTime
      + 0.5 * acc * secondaryTime * secondaryTime;
  }
  return result;
}
```

Example: Split Temporary Variable

```
double getDistanceTravelled(int time) {
  double result = 0;
  final double primaryAcc = primaryForce / mass;
  int primaryTime = Math.min(time, delay);
  result = 0.5 * primaryAcc * primaryTime * primaryTime;
  int secondaryTime = time – delay;
  if (secondaryTime > 0) {
    double primaryVelocity = acc * _delay;
    double acc = (primaryForce + secondaryForce) / mass;
    result += primaryVelocity * secondaryTime
      + 0.5 * acc * secondaryTime * secondaryTime;
  }
  return result;
}
```

Example: Split Temporary Variable

```
double getDistanceTravelled(int time) {
  double result = 0;
  final double primaryAcc = primaryForce / _mass;
  int primaryTime = Math.min(time, delay);
  result = 0.5 * primaryAcc * primaryTime * primaryTime;
  int secondaryTime = time – delay;
  if (secondaryTime > 0) {
    double primaryVelocity = acc * delay;
    final double secondaryAcc = (primaryForce + secondaryForce) / mass;
    result += primaryVelocity * secondaryTime
      + 0.5 * secondaryAcc * secondaryTime * secondaryTime;
  }
  return result;
}
```

Inline Method

Name
Inline Method

Summary
A method body is as clear as its name

Goal
Inline the method's body into its callers

Mechanics
- check if method is not polymorphic
- find all calls of the method
- replace them with body
- compile & test
- remove method definition

Example: Inline Method

Example 1

```
public class Worker {
   public Money getSalary() {
      return isManger() ? 10000 : 2000;
   }
   private isManager() {
      return job == 'M';
   }
}
```

Example: Inline Method

Example 1

```
public class Worker {
   public Money getSalary() {
      return (job == 'M') ? 10000 : 2000;
   }
}
```

Example: Inline Method

Example 2

```
public class Worker {
   public Money getSalary() {
      if (job == 'M') {
         return 10000;
      } else if (employmentYears > 10) {
         return 2000 + 0.5 * employmentYears;
      } else {
         return 2000 + 0.3 * (employmentYears – 3);
      }
   }
}

System.out.println("Salary is: " + worker.getSalary());
```

Encapsulate Downcast

Name
Encapsulate Downcast

Summary
The method returns an object that needs to de downcasted by its callers

Goal
Move the downcast inside the method

Mechanics
- look for cases (e.g. collection-related) in which you have to downcast the result from calling a method
- move the downcast into the method

Example: Encapsulate Downcast

Example

```
public class Lecture {
   List students = new ArrayList();
   Object lastStudent() {
     return students.lastElement();
   }
}

Student last = (Student) lecture.lastStudent();
```

Example: Encapsulate Downcast

Example

```
public class Lecture {
   List students = new ArrayList();
   Student lastStudent() {
      return (Student) students.lastElement();
   }
}
```

```
public class Lecture {
   List<Student> students = new ArrayList<Student>();
   Student lastStudent() {
      return students.lastElement();
   }
}
```

Catalog of software refactorings

Related code smells

- data class
- lazy class
- large class
- inappropriate intimacy

Self-Encapsulate Field

Name
Self-Encapsulate Field

Summary
A directly accessed field has awkward couplings or needs overriding

Goal
Provide accessors to access the field

Mechanics
- create (private/protected) getter and setter for the field
- replace references to the field with appropriate calls to accessors
- make the field private
- compile & test

Encapsulate Field

Name
Encapsulate Field

Summary
An public field is exposed to clients

Goal
Make it private and provide public accessors

Mechanics
- create (public) getter & setter
- replace refereces to the field with calls to the accessors
- compile & test
- declare field as private
- compile & test

Encapsulate Collection

Name
Encapsulate Collection

Summary
A method returns a collection

Goal
Provide a read-only view to a collection

Mechanics
- add an *add()* and *remove()* methods for the collection
- modify the setter to use the *add()*/*remove()*
- compile & test
- modify the users of getting method so that they use *add()*/*remove()*
- make the getter to return a read-only view of the collection
- compile & test
- (make the method to return Iterator instead of Collection)

Example: Encapsulate Collection

Example

```
public class Student {
   Collection lectures;

   public Collection getLectures() {
      return lectures;
   }
}
```

Example: Encapsulate Collection

```
public class Student {
   Collection lectures;

   public Collection getLectures() {
      return Collections.unmodifiableCollection(lectures);
   }
}
```

```
public class Student {
   Collection lectures;

   public Iterator getLectures() {
      return lectures.iterator();
   }
}
```

Move Field

Name
Move Field

Summary
A field is mostly used by a foreign class

Goal
Transfer the field to the target class

Mechanics
- encapsulate the field if it is public
- create a field in the target class with setter and getter provided
- compile the target class
- determine how to reference the target object from the source
- remove the field from the source class
- replace references to the source field with calls to the appropriate method on the target
- compile & test

Example: Move Field

Example

```
class Account {
   private AccountType _type;
   private double _interestRate;

   double interestForAmountInDays(double amount, int days) {
      return _interestRate * amount * days / 365;
   }
}
```

Example: Move Field

```
class AccountType {
   private double _interestRate;

   void setInterestRate(double rate) {
      _interestRate = rate;
   }
   double getInterestRate() {
      return _interestRate;
   }
}
```

```
class Account {
   private AccountType _type;
   private double _interestRate;

   double interestForAmountInDays(double amount, int days) {
      return _type.getInterestRate() * amount * days / 365;
   }
}
```

Move Method

Name
Move Method

Summary
A method is using more features of another class than its own

Goal
Create a new method in the target class.

Mechanics
- check if the method is not polymorphic in source class hierarchy
- declare the method in the target class
- copy the code of the method over to the target class
- compile the target class
- determine how to reference the target object from the source (either by reference or passing a method parameter)
- turn the source method into a delegation
- compile & test

Example: Move Method

Example

```
class Account {
   private AccountType _type;
   private int _daysOverdrawn;

   double overdraftCharge() {
      if (_type.isPremium() {
         double result = 10;
         if (_daysOverdrawn > 7) result += (_daysOverdrawn – 7) * 0.85;
         return result;
      } else {
         return _daysOverdrawn * 1.75;
      }
   }
   double bankCharge() {
      double result = 4.5;
      if (_daysOverdrawn > 0) result += overdrawftCharge();
      return result;
   }
}
```

Example: Move Method

```
class AccountType {
   double overdraftCharge(int _daysOverdrawn) {
      if (isPremium()) {
         double result = 10;
         if (daysOverdrawn > 7) result += (daysOverdrawn – 7) * 0.85;
         return result;
      } else {
         return daysOverdrawn * 1.75;
      }
   }
}
```

```
class Account {
   double overdraftCharge() {
      return _type.overdraftCharge(_daysOverdrawn);
   }
}
```

Example: Move Method

```
class AccountType {
   double overdraftCharge(Account account) {
      if (isPremium()) {
         double result = 10;
         if (account.getDaysOverdrawn() > 7)
            result += (account.getDaysOverdrawn() – 7) * 0.85;
         return result;
      } else {
         return account.getDaysOverdrawn() * 1.75;
      }
   }
}
```

Example: Move Method

```
class AccountType {
   double overdraftCharge(int _daysOverdrawn) {
      if (isPremium()) {
         double result = 10;
         if (daysOverdrawn > 7) result += (daysOverdrawn – 7) * 0.85;
         return result;
      } else {
         return daysOverdrawn * 1.75;
      }
   }
}
```

Catalog of software refactorings

Related code smells
- primitive obsession
- switch statements

Replace Error Code with Exception

Name
Replace Error Code with Exception

Summary
A method signals an error by setting specific error code

Goal
Throw an exception instead

Mechanics
- consider checked vs. unchecked exceptions
- create a new method (with different name!) that throws the exception
- modify the old one to use the new one
- compile & test
- make callers to call the new method, compile & test
- delete the old one

Example: Replace Error Code with Exception

Example

```
public class Account {
   private int _balance;

   public int withdraw(int amount) {
      if (amount > _balance) {
         return –1;
      } else {
         _balance –= amount;
         return 0;
      }
   }
}
```

Example: Replace Error Code with Exception

Example
```
public class Account {
   private int _balance = 0;

   public int withdraw(int amount) {
      try {
        withdrawEx(amount);
      } catch (BalanceException) {
        return –1;
      }
     return 0;
   }
```
```
public void withdrawEx(int amount)
throws BalanceException {
   if (amount > _balance) {
      throw new BalanceException(
           "Sorry");
   }
   _balance –= amount;
  }
}
```

Replace Exception with Test

Name
Replace Exception with Test

Summary
An exception is thrown on a condition that could be checked first

Goal
Check the condition first and (possibly) fail fast

Mechanics
- put an up-front test and copy the *catch* code to appropriate branch of statement
- add an assertion in *catch* clause to notify the clause execution
- compile & test
- remove *catch* and *try* clauses
- compile & test

Example: Replace Exception with Test

Example
```
Stack<Connection> pool;
Stack<Connection> allocated;

public Connection getConnection() {
   Connection conn;
   try {
      conn = pool.pop();
   } catch (EmptyStackException ex) {
      conn = new DriverManager.getConnection(url, user, password);
   }
   allocated.push(conn);

   return conn;
}
```

Example: Replace Exception with Test

```
Stack<Connection> pool;
Stack<Connection> allocated;

public Connection getConnection() {
   Connection conn;
   if (pool.isEmpty()) {
      conn = new DriverManager.getConnection(url, user, pass);
   } else {
      conn = pool.pop();
   }
   allocated.push(conn);

   return conn;
}
```

Replace Constructor with Factory Method

Name
Replace Constructor with Factory Method

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

Goal
Replace the constructor with Factory Method pattern

Mechanics
- create a factory method; make its body a call to the constructor
- replace calls to constructor with calls to the factory method
- compile & test
- declare the constructor private
- compile

Examples

Example: Replace Constructor with Factory Method

Example
```
public class Person {
   public Person(String name) {
      //...
   }
}
```
```
public class Person {
   private Person(String name) {
      //...
   }
   public static Person getInstance(String name) {
     return new Person(name);
   }
}
```

Introduce Null Object

Name
Introduce Null Object

Summary
There are repeated checks for a null value

Goal
Replace the null value with an object representing null

Mechanics
- create a subclass of the source class to act as a null version of the class
- create *isNull()* operation on the source class and the null subclass
- compile
- replace all places that give out a null to return a null object instead
- replace all checks for null with calls to *isNull()* method
- compile & test
- override the operation performed by clients if null/not null is found
- remove the condition check for clients that use the overriden behavior

Example: Introduce Null Object

Example

```
Student student = //...
String name = null

if (student != null) {
   name = student.getName();
} else {
   name = "";
}

public class Student {
   public String getName() {
      return name;
   }
}
```

Example: Introduce Null Object

```
Student student = //...

String name = student.getName();

public class NullStudent extends Student {
   public String getName() {
     return "";
   }
}
```

Q&A