Advanced Object-Oriented Design
Lecture 7

Software Refactoring
Complex transformations

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

Agenda

Refactorings related to
- feature envy
- message chains
- inappropriate intimacy
- middle man

Hide Delegate

Name
Hide Delegate

Summary
A client is calling a delegate class of an object

Goal
Create methods on the server to hide the delegate

Mechanics
- for each method on the delegate, create a simple delegating method on the server
- adjust the client to call the server, compile & test
- remove the server's accessors for the delegate
- compile & test

Example: Hide Delegate

```
public class Person {
   private Department _department;;

   public Department getDepartment() {
      return _department;
   }

   public void setDepartment(Department department) {
      _department = department;
   }
}

public class Department {
   private String _chargeCode;
   private Person _manager;

   public Department(Person manager) {
      _manager = manager;
   }

   public Person getManager() {
      return _manager;
   }
}
```

example by M. Fowler

```
Person manager = john.getDepartment().getManager();
```

Example: Hide Delegate

```
public class Person {
   private Department _department;;

   public void setDepartment(Department department) {
      _department = department;
   }

   public Person getManager() {
      return _department.getManager();
   }
}
```

```
Person manager = john.getManager();
```

example by M. Fowler

Remove Middle Man

Name
Remove Middle Man

Summary
A class is doing too much simple delegation

Goal
Get the client to call the delegate directly

Mechanics
- create an accessor for the delegate
- remove selected methods from the server and replace them in the client with a call to the delegate
- compile & test

Agenda

Refactorings related to

- lazy class
- large class
- inappropriate intimacy

Change Value to Reference

Name
Change Value to Reference

Summary
A class has many equal instances that can be replaced by one

Goal
Turn the class into a reference object

Mechanics

- replace the constructor with *factory method*
- decide which object is responsible for providing access to the objects
- decide whether objects are precreated or created on the fly
- alter the factory method to return the reference object
- compile & test

Example: Change Value to Reference

```
public class Customer {
   private final _name = name;

   public Customer(String name) {
      _name = name;
   }
}

class Order {
   Customer _customer;

   public Order(String customerName) {
      _customer = new Customer(customerName);
   }

   public void setCustomer(String customerName) {
      _customer = new Customer(customerName);
   }
}
```

example by M. Fowler

Example: Change Value to Reference

```
public class Customer {
   private final String name;

   private Customer(String name) {
      _name = name;
   }

   public static Customer create(String name) {
      return new Customer(name);
   }

}

class Order {
   Customer _customer;

   public Order(String customerName) {
      _customer = Customer.create(customerName);
   }

   public void setCustomer(String customerName) {
      _customer = Customer.create(customerName);
}
```

example by M. Fowler

Example: Change Value to Reference

```
public class Customer {
   private final String name;
   private static Map<String, Customer> customers =
      new HashMap<String, Customer>();

   private Customer(String name) {
      name = name;
   }

   public static Customer create(String name) {
      Customer customer = customers.get(name);

      if (customer == null) {
         customer = new Customer(name);
         customers.put(name, customer);
      }

      return customer;
   }
}
```

example by M. Fowler

Change Reference to Value

Name
Change Reference with Value

Summary
A reference object is small and immutable

Goal
Turn it into a value object

Mechanics

- check if the candidate object is immutable or can be one
- override the *equals()* and *hashCode()* methods
- compile & test
- consider removing factory method and making the constructor public

Example: Change Reference to Value

```
public class Currency {
    private String _code;
    private static Map<String, Currency> codes =
        new HashMap<String, Currency>();

    private  Currency (String code) {
        _code = code;
    }

    public String getCode() {
        return _code;
    }

    public static Currency get(String code) {
        Currency curr = codes.get(code);
        if (curr == null) {
            curr = new Currency(code);
            codes.put(code, curr);
        }

        return curr;
    }
}
```

example by M. Fowler

Example: Change Reference to Value

```
public class Currency {
    public boolean equals(Object arg) {
        Currency other = (Currency) arg;
        return _code.equals(other._code);
    }

    public int hashCode() {
       return _code.hashCode();
    }

    public Currency(String code) {
        _code = code;
    }
}
```

```
assertEquals(new Currency("USD"), new Currency("USD")); // true
```

```
assertSame(new Currency("USD"), new Currency("USD")); // ?????
```

example by M. Fowler

Change Unidirectional Association to Bidirectional

Name

Change Unidirectional Association to Bidirectional

Summary

Two classes need to know about each other

Goal

Add back pointers and change modifiers to update both sets

Mechanics

- add a field for back pointer
- choose a controller for the association
- create a helper method on controlled side, name it appropriately
- if the existing modifier is on controlling side, make it update back pointer
- if it is not, create a controlling method on controlling side and call it from exisiting modifiers

Example: Change Unidirectional Association to Bidirectional

```
public class Order {
    private Customer _customer;

    Customer getCustomer() {
        return _customer;
    }

    void setCustomer(Customer customer) {
        _customer = customer;
    }
}

public class Customer {
    private Set _orders = new HashSet();

    Set friendOrders() {
        return _orders;
    }
}
```

example by M. Fowler

Example: Change Unidirectional Association to Bidirectional

```
public class Order {
    private Customer _customer;

    Customer getCustomer() {
        return _customer;
    }

    void setCustomer(Customer customer) {
        if (_customer != null){
            _customer.friendOrders().remove(this);
        }

        _customer = customer;

        if (_customer != null) {
            _customer.friendOrders().add(this);
        }
    }
}

public class Customer {
    void addOrder(Order order) {
        order.setCustomer(this);
    }
}
```

example by M. Fowler

Example: Change Unidirectional Association to Bidirectional

```
public class Order {
    Set _customers = new HashSet();

    void addCustomer(Customer customer) {
        customer.friendOrders().add(this);
        _customers.add(customer);
    }

    void removeCustomer(Customer customer) {
        customer.friendOrders().remove(this);
        _customers.remove(customer);
    }
}

public class Customer {
    Set _orders = new HashSet();

    void addOrder(Order order) {
        order.addCustomer(this);
    }

    void removeOrder(Order order) {
        order.removeCustomer(this);
    }
}
```

example by M. Fowler

Change Bidirectional Association to Unidirectional

Name
Change Bidirectional Association to Unidirectional

Summary
There is a two-way association, but one class no longer needs access to the other

Goal
Put the method's body into its callers, remove the method

Mechanics
- if clients need to use the getter, self-encapsulate the field and adjust the getter
- if they do not, change them so that they get the object in another way
- remove all updated to the field, remove the field
- compile & test

Example: Change Bidirectional Association to Undirectional

```
public class Order {
    private Customer _customer;

    Customer getCustomer() {
        return _customer;
    }
    void setCustomer(Customer customer) {
        if (_customer != null) _customer.friendOrders().remove(this);
        _customer = customer;
        if (_customer != null) _customer.friendOrders().add(this);
    }
    double getDiscountedPrice() {
        return _customer.getDiscount() * 0.9;
    }
}

public class Customer {
    private Set _orders = new HashSet();

    Set friendOrders() {
        return _orders;
    }
    void addOrder(Order order) {
       order.setCustomer(this);
    }
}
```

example by M. Fowler

Example: Change Bidirectional Association to Undirectional

```
public class Order {

    double getDiscountedPrice(Customer customer) {
        return customer.getDiscount() * 0.9;
    }
}

public class Customer {
    private Set _orders = new HashSet();

    Set friendOrders() {
        return _orders;
    }
    void addOrder(Order order) {
       order.setCustomer(this);
    }
}
```

example by M. Fowler

Replace Inheritance with Delegation

Name
Replace Inheritance with Delegation

Summary
A subclass does not want to inherit data

Goal
Create a field for superclass, delegate calls to it and remove subclassing

Mechanics
- create a field for superclass, initialize it with this
- change methods in subclass to use the delegate field
- remove the subclass declaration, replace the delegate assignment with the assignment to a new object
- add simple delegations for all superclass methods used
- compile & test

Example: Replace Inheritance with Delegation

```
public class MyStack extends Vector {

   public void push(Object object) {
      insertElementAt(object, 0);
   }

   public Object pop() {
      Object result = super.firstElement();
      super.removeElementAt(0);

      return result;
   }
}
```

example by M. Fowler

Example: Replace Inheritance with Delegation

```
public class MyStack extends Vector {
   private Vector _vector = this;

   public void push(Object object) {
      _vector.insertElementAt(object, 0);
   }

   public Object pop() {
      Object result = _vector.firstElement();
      _vector.removeElementAt(0);
      return result;
   }
}
```

example by M. Fowler

Example: Replace Inheritance with Delegation

```
public class MyStack {                          // no inheritance
   private Vector _vector = this;

   public void push(Object object) {
      insertElementAt(object, 0);
   }

   public Object pop() {
      Object result = _vector.firstElement();
      _vector.removeElementAt(0);
      return result;
   }

   public int size() {
      return _vector.size();
   }

   public boolean isEmpty() {
      return _vector.isEmpty();
   }
}
```

example by M. Fowler

Replace Delegation with Inheritance

Name
Change Delegation with Inheritance

Summary
There are many simple delegations for the entire interface

Goal
Make the delegating class a subclass of the delegate

Mechanics
- make the delegating object a subclass of the delegate, compile
- set the delegate field to the object itself
- replace the delegations to superclass with calls to the object itself
- compile & test
- remove the delegate field

Example: Replace Delegation with Inheritance

```
public class Employee {
   private Person _person = new Person();

   public String getName() {
      return _person.getName();
   }

   public void setName(String name) {
      return _person.setName(name);
   }
}

public class Person {
   private String _name;

   public String getName() {
      return _name;
   }
}
```

example by M. Fowler

Example: Replace Delegation with Inheritance

```
public class Employee extends Person {
   private Person _person = this;

   public String getName() {
      return this.getName();
   }

   public void setName(String name) {
      return _person.setName(name);
   }
}

public class Person {
   private String _name;

   public String getName() {
      return _name;
   }
}
```

example by M. Fowler

Example: Replace Delegation with Inheritance

```
public class Employee extends Person {

}

public class Person {
   private String _name;

   public String getName() {
      return _name;
   }
}
```

example by M. Fowler

Agenda

Refactorings related to
- switch statements
- complicated boolean expressions
- simulated inheritance

Decompose Conditional

Name
Decompose Conditional

Summary
There is a complicated conditional statement

Goal
Extract method from the condition, *then* part and *else* parts

Mechanics
- extract the condition into its own method
- extract the *then* part and the *else* part into their own methods.

Example: Decompose Conditional

```
if (date.before(SUMMER_START) || date.after(SUMMER_END) {
    charge = quantity * winterRate + winterServiceCharge;
} else {
    charge = quantity * summerRate;
}
```

```
if (notSummer(date)) {
    charge = winterCharge(quantity);
} else {
    charge = summerCharge(quantity);
}

private boolean notSummer(Date date) {
    return date.before(SUMMER_START) || date.after(SUMMER_END);
}

private double summerCharge(int quantity) {
    return quantity * winterRate + winterServiceCharge;
}

private double winterCharge(int quantity) {
    return quantity * summerRate;
}
```

example by M. Fowler

Reverse Conditional

Name
Reverse Conditional

Summary
A conditional would be easier to understand if reversed.

Goal
Reverse the sense of the conditional and reorder its clauses

Mechanics
- remove negative from conditional (apply deMorgans' Law if necessary)
- switch clauses
- compile & test

Example: Reverse Conditional

```
public void someMethod() {
   if (! isCommited() && ! ((index > 0) || (str.indexOf('!') > -1))){
      // do something...
   } else {
      // do something else...
   }
}
```

```
public void someMethod() {
   if (! (isCommited() || ((index > 0) || (str.indexOf('!') > -1))) {
      // do something...
   } else {
      // do something else...
   }
}
```

```
public void someMethod() {
   if ((isCommited() || (index > 0) || (str.indexOf('!') > -1)) {
      // do something else...
   } else {
      // do something...
   }
}
```

Replace Nested Conditional with Guard Clauses

Name
Replace Nested Conditional with Guard Clauses

Summary
Method has conditional behavior that makes the normal path of execution unclear

Goal
Use guard clauses for all the special cases

Mechanics
- for each check put it in the front guard clause, which returns from the method or throws an exception
- compile and test after each change

Example: Replace Nested Conditional with Guard Clauses

```
double getPayAmount() {
    double result = 0.0;

    if (_isDead) {
        result = deadAmount();
    } else {
        if (_isSeparated) {
            result = separatedAmount();
        } else {
            if (_isRetired) {
                result = retiredAmount();
            } else {
                result = normalPayAmount();
            }
        }
    }

    return result;
}
```

example by M. Fowler

Example: Replace Nested Conditional with Guard Clauses

```
double getPayAmount() {
    double result = 0.0;

    if (_isDead) return deadAmount();

    if (_isSeparated) {
        result = separatedAmount();
    } else {
        if (_isRetired) {
            result = retiredAmount();
        } else {
            result = normalPayAmount();
        }
    }

    return result;
}
```

example by M. Fowler

Example: Replace Nested Conditional with Guard Clauses

```
double getPayAmount() {

    if (_isDead) return deadAmount();
    if (_isSeparated) return separatedAmount();
    if (_isRetired) return retiredAmount();

    return normalPayAmount();
}
```

example by M. Fowler

Consolidate Conditional Expressions

Name
Consolidate Conditional Expression

Summary
There is a sequence of conditional tests with same result

Goal
Combine them into a single expression and extract it

Mechanics
- check that none of the conditionals has side effects
- replace the string of conditionals with a single conditional using logical operators
- compile & test
- (extract expression as a method)

Example: Consolidate Conditional Expressions

```
if (isSpecialDeal()) {
    total = price * 0.95;
    send();
} else {
    total = price * 0.98;
    send();
}
```

```
if (isSpecialDeal()) {
    total = price * 0.95;
} else {
    total = price * 0.98;
}

send();
```

example by M. Fowler

Remove Control Flag

Name
Remove Control Flag

Summary
A variable is acting as a control flag for a serie of conditionals

Goal
Use a *break* or *return* statement instead

Mechanics
- find the value of the flag that makes you to leave out of the conditional
- replace assignments of the value with a *break* or *continue*
- compile and test

Example: Remove Control Flag

```
void checkSecurity(String[] people) {
   boolean found = false;
   for (int i = 0; i < people.length; i++) {
      if (!found) {
         if (people[i].equals("Don")) {
            sendAlert();
            found = true;
         }
         if (people[i].equals("John")) {
            sendAlert();
            found = true;
         }
      }
   }
}
```

example by M. Fowler

Example: Remove Control Flag

```
void checkSecurity(String[] people) {
   boolean found = false;
   for (int i = 0; i < people.length; i++) {
      if (!found) {
         if (people[i].equals("Don")) {
            sendAlert();
            break;
         }
         if (people[i].equals("John")) {
            sendAlert();
            break;
         }
      }
   }
}
```

example by M. Fowler

Example: Remove Control Flag

```
void checkSecurity(String[] people) {
   for (int i = 0; i < people.length; i++) {
      if (people[i].equals("Don")) {
         sendAlert();
         break;
      }
      if (people[i].equals("John")) {
         sendAlert();
         break;
      }
   }
}
```

example by M. Fowler

Replace Type Code with Class

Name
Replace Type Code with Class

Summary
A class has a numeric type code that does not affect its behaviour

Goal
Replace the member with a new class

Mechanics
- create a new class for the code
- modify the implementation of the source class to use the new class
- compile & test
- for each method on the source class that uses the code, create a new method that uses the new class instead
- change the clients so that they use the new interface, compile & test
- remove the old interface and static declarations of the codes

Example: Replace Type Code with Class

```
public class Person {
   public static final int O = 0;
   public static final int A = 1;
   public static final int B = 2;
   public static final int AB = 3;

   private int _bloodGroup;

   public Person(int bloodGroup) {
      _bloodGroup = bloodGroup;
   }

   public void setBloodGroup(int bloodGroup) {
      _bloodGroup = bloodGroup;
   }

   public int getBloodGroup() {
      return _bloodGroup;
   }
}
```

example by M. Fowler

Example: Replace Type Code with Class

```
public class BloodGroup {
   public static final BloodGroup O = new BloodGroup(0);
   public static final BloodGroup A = new BloodGroup(1);
   public static final BloodGroup B = new BloodGroup(2);
   public static final BloodGroup AB = new BloodGroup(3);
   private static final BloodGroup[] _values = {O, A, B, AB};

   private int _code;

   public BloodGroup(int code) {
      _code = code;
   }

   public static BloodGroup code(int index) {
      return _values[code];
}
```

example by M. Fowler

Example: Replace Type Code with Class

```
public class Person {
   public static final int O = BloodGroup.0.getCode();
   public static final int A = BloodGroup.A.getCode();
   public static final int B = BloodGroup.B.getCode();
   public static final int AB = BloodGroup.AB.getCode();

   private BloodGroup_bloodGroup;

   public Person(int bloodGroupCode) {
      _bloodGroup = BloodGroup.code(bloodGroupCode);
   }
   public Person(BloodGroup bloodGroup) {
      _bloodGroup = bloodGroup;
   }

   public void setBloodGroup(int bloodGroup) {
      _bloodGroup = bloodGroup;
   }

   public BloodGroup getBloodGroup() {
      return __bloodGroup();
   }
   public int getBloodGroupCode() {
      return __bloodGroup.getCode();
   }
}
```

example by M. Fowler

Example: Replace Type Code with Class

```
public class Person {
   // blood group codes removed

   private BloodGroup_bloodGroup;

   public Person(int bloodGroupCode) {
      _bloodGroup = BloodGroup.code(bloodGroupCode);
   }
   public Person(BloodGroup bloodGroup) {
      _bloodGroup = bloodGroup;
   }

   public void setBloodGroup(int bloodGroup) {
       _bloodGroup = bloodGroup;
   }

   public BloodGroup getBloodGroup() {
       return __bloodGroup();
   }
   public int getBloodGroupCode() {
       return __bloodGroup.getCode();
   }
}
```

example by M. Fowler

Replace Type Code with Subclasses

Name
Replace Type Code with Subclasses

Summary
There is immutable type code that affects behavior of class

Goal
Replace the type code with subclasses

Mechanics

- self-encapsulate the type code
- for each value of the type code, create a subclass; override the getter of the type code in the subclass to return the relevant value
- compile & test
- remove type code field from superclass; declare the accessors for type code as abstract
- compile & test

Example: Replace Type Code with Subclasses

```
public class Employee {
   public static final int ENGINEER = 0;
   public static final int SALESMAN = 1;
   public static final int MANAGER = 2;

   private int _type;

   public Employee(int type) {
       _type = type;
   }

   public int getType() {
      return _type;
   }
}
```

example by M. Fowler

Example: Replace Type Code with Subclasses

```
public class Employee {
   public static final int ENGINEER = 0;
   public static final int SALESMAN = 1;
   public static final int MANAGER = 2;

   private int _type;

   private Employee(int type) {
   _type = type;
   }

   static Employee create(int type) {
      return new Employee(type);
   }

   public int getType() {
      return _type;
   }
}
```

example by M. Fowler

Example: Replace Type Code with Subclasses

```
public class Engineer extends Employee {
   public int getType() {
      return Employee.ENGINEER;
   }
}
```

example by M. Fowler

Example: Replace Type Code with Subclasses

```
public class Employee {

   static Employee create(int type) {
      if (type == Employee.ENGINEER) {
         return new Engineer();
      }

      return new Employee(type);
   }
}
```

example by M. Fowler

Example: Replace Type Code with Subclasses

```
public class Employee {

   abstract int getType();

   static Employee create(int type) {
      switch (type) {
         case Employee.ENGINEER:
             return new Engineer();
             break;
         case Employee.SALESMAN:
             return new Salesman();
             break;
         case Employee.MANAGER:
             return new Manager();
             break;
         default: throw new IllegalArgumentException();
      }
   }
}
```

example by M. Fowler

Replace Type Code with State

Name
Replace Type Code with State

Summary
Type code affects the behavior of class, but subclassing cannot be used

Goal
Replace the type code with a state object

Mechanics

- self-encapsulate the type code
- create a new abstract class for the type codes – the state object
- add subclasses of the state object, one for each type code
- create and override type code queries in the subclasses
- compile
- create a field in the old class for the new state object
- adjust the type code setters in the original class to assign an instance of appropriate state object subclass; compile & test

Example: Replace Type Code with State

```
public class Employee {
   public static final int ENGINEER = 0;
   public static final int SALESMAN = 1;
   public static final int MANAGER = 2;

   private int _type;

   public Employee(int type) {
       _type = type;
   }

   public int getType() {
       return _type;
   }

   public int payAmount() {
      switch (_type) {
         case ENGINEER: return _monthlySalary;
         case SALESMAN: return _monthlySalary + _commission;
         case MANAGER:  return _monthlySalary + _bonus;
      }
   }
}
```

example by M. Fowler

Example: Replace Type Code with State

```
abstract class EmployeeType {
    public abstract int getTypeCode();
}

public class Engineer extends EmployeeType {
    public int getTypeCode(){
        return Employee.ENGINEER;
    }
}

public class Manager extends EmployeeType {
    public int getTypeCode(){
        return Employee.MANAGER;
    }
}

public class Salesman extends EmployeeType {
    public int getTypeCode(){
        return Employee.Salesman;
    }
}
```

example by M. Fowler

Example: Replace Type Code with State

```
public class Employee {
   public static final int ENGINEER = 0;
   public static final int SALESMAN = 1;
   public static final int MANAGER = 2;

   private EmployeeType _type;

   public Employee(int type) {
       _type = type;
   }

   public int getType() {
       return __type.getTypeCode();
   }

   public void setType(int type) {
       switch (type) {
           _case ENGINEER: _type = new Engineer(); break;
           _case SALESMAN: _type = new Salesman(); break;
           _case MANAGER : _type = new Manager(); break;
       }
   }
}
```

example by M. Fowler

Example: Replace Type Code with State

```
public class Employee {
   void setType(int type) {
      _type = EmployeeType.newType(type);
   }
   public int payAmount() {
      switch (getType()) {
         case ENGINEER: return _monthlySalary;
         case SALESMAN: return _monthlySalary + _commission;
         case MANAGER:  return _monthlySalary + _bonus;
      }
   }
}

public abstract class EmployeeType {
   public static final int ENGINEER = 0;
   public static final int SALESMAN = 1;
   public static final int MANAGER = 2;

   public static EmployeeType newType (int type) {
       switch (type) {
           _case ENGINEER: _type = new Engineer(); break;
           _case SALESMAN: _type = new Salesman(); break;
           _case MANAGER : _type = new Manager(); break;
       }
   }
}
```

example by M. Fowler

Replace Conditional with Polymorphism

Name
Replace Conditional with Polymorphism

Summary
Conditional chooses different behavior depending on the type of an object

Goal
Move each leg of conditional to an overriding method in a subclass

Mechanics
- extract and pull up the conditional at the top of inheritance hierarchy
- create abstract method in the superclass
- override the method in subclasses, copying there appropriate leg of the conditional
- compile & test
- remove the copied leg
- compile & test

Example: Replace Conditional with Polymorphism

```
public class Employee {
   private EmployeeType _type;

   public int payAmount() {
      switch (getTypeCode()) {
         case ENGINEER: return _monthlySalary;
         case SALESMAN: return _monthlySalary + _commission;
         case MANAGER:  return _monthlySalary + _bonus;
      }
   }

   public int getTypeCode() {
       return _type.getTypeCode();
   }
}

public abstract class EmployeeType {
   public abstract int getTypeCode();
}

public Engineer extends EmployeeType {
   public int getTypeCode() {
      return EmployeeType.ENGINEER;
   }
}
```

example by M. Fowler

Example: Replace Conditional with Polymorphism

```
public class Employee {
   private EmployeeType _type;

   public int payAmount() {
      return _type.payAmount(this);
   }
}

public abstract class EmployeeType {
   public int payAmount(Employee employee) {
      switch (getTypeCode()) {
         case ENGINEER:
            return emp.getMonthlySalary();
         case SALESMAN:
            return emp.getMonthlySalary() + getCommission();
         case MANAGER:
            return emp.getMonthlySalary() + getBonus();
      }
   }
}
```

example by M. Fowler

Example: Replace Conditional with Polymorphism

```
public abstract class EmployeeType {
   abstract int payAmount(Employee employee);
}

public class Engineer extends EmployeeType {
   public int payAmount(Employee emp) {
       return emp.getMonthlySalary();
   }
}

public class Salesman extends EmployeeType {
   public int payAmount(Employee emp) {
       return emp.getMonthlySalary() + emp.getCommission();
   }
}

public class Manager extends EmployeeType {
   public int payAmount(Employee emp) {
       return emp.getMonthlySalary() + emp.getBonus();
   }
}
```

example by M. Fowler

Replace Conditional with Visitor

Name
Replace Conditional with Visitor

Summary
An "aggressive" conditional chooses different behaviour depending on the type of an object and repeats itself in a large number throughout the code

Goal
Turn the conditional legs into visitable objects and visit them

Mechanics
- Create concrete instance of *Visitable* object for each data type in conditional
- Create concrete instance of *Visitor* that encapsulates logic of each conditional
- Visit *Visitable* by *Visitor*

Agenda

Refactorings related to
- refused bequest
- inappropriate intimacy
- lazy class
- large class
- parallel inheritance hierarchies

Pull Up Field/ Method

Name
Pull Up Field/Method

Summary
Two subclasses have the same field/method

Goal
Move the field/method to the superclass

Mechanics
- inspect the fields/methods to ensure they are identical
- create a new method in superclass, copy the body of one of methods, adjust and compile
- delete one subclass method, compile & test
- proceed with deleting subclass methods

Pull Up Constructor Body

Name
Pull Up Constructor Body

Summary
Constructors on subclasses are almost identical

Goal
Create a superclass constructor, call it from subclasses

Mechanics
- define a superclass constructor
- move the common code from the subclass to the superclass
- call the superclass constructor as first step in the subclass constructor
- compile & test

Example 1: Pull Up Constructor Body

```
public class Employee {
    protected String _name;
    protected String _id;
}

public class Manager extends Employee {
    private int _grade;

    public Manager (String name, String id, int grade) {
        _name = name;
        _id = id;
        _grade = grade;
    }
}
```

example by M. Fowler

Example 1: Pull Up Constructor Body

```
public class Employee {
    protected String _name;
    protected String _id;

    protected Employee(String name, String id) {
        _name = name;
        _id = id;
    }
}

public class Manager extends Employee {
    private int _grade;

    public Manager (String name, String id, int grade) {
        super(name, id);
       _grade = grade;
    }
}
```

example by M. Fowler

Example 2: Pull Up Constructor Body

```
public class Employee {
    boolean isPriviliged() {...}
    void assignCar() {...}
}

public class Manager extends Employee {
    public Manager (String name, String id, int grade) {
        super(name, id);
       _grade = grade;
        if (isPriviliged()) {
            assignCar();
        }
    }

    boolean isPriviliged() {
        return _grade > 4;
    }
}
```

example by M. Fowler

Example 2: Pull Up Constructor Body

```
public class Employee {
    boolean isPriviliged() {...}
    void assignCar() {...}
    void initialize() {
       if (isPriviliged()) {
           assignCar();
       }
    }
}

public class Manager extends Employee {
    public Manager (String name, String id, int grade) {
        super(name, id);
       _grade = grade;
        initialize();
    }

    boolean isPriviliged() {
        return _grade > 4;
    }
}
```

example by M. Fowler

Push Down Field/ Method

Name
Push Down Field/Method

Summary
Behavior on superclass is relevant only for some subclasses

Goal
Move it to those subclasses

Mechanics
- declare a member in all subclasses and copy the body into each subclass (make them public/protected)
- remove method from superclass (or declare abstract)
- compile & test
- remove method from subclasses that do not need it

example by M. Fowler

Extract Interface

Name
Extract Interface

Summary
Some clients use the same subset of class's interface

Goal
Extract the subset into an interface

Mechanics
- create an empty interface
- declare the common operations in the interface
- declare the relevant classes as implementing the interface
- adjust the client type declarations to use the interface

Example: Extract Interface

```
double charge(Employee emp, int days) {
    int base = emp.getRate() * days;
    if (emp.hasSpecialSkill()) {
        return base * 1.05;
    } else {
        return base;
    }
}
```

```
interface Billable {
    public int getRate();
    public boolean hasSpecialSkill();
}

class Employee implements Billable {
}

double charge(Billable emp, int days) {
    int base = emp.getRate() * days;
    if (emp.hasSpecialSkill()) {
        return base * 1.05;
    } else {
        return base;
    }
}
```

example by M. Fowler

Extract Superclass

Name
Extract Superclass

Summary
There are two classes with similar features

Goal
Move the common features to a newly created superclass

Mechanics
- create a blank abstract superclass
- pull up fields, whole methods and constructor body
- compile & test at every change
- if necessary, split remaining methods and pull them up or *form template method*
- change references in clients to superclass (if applicable)

Extract Subclass

Name
Extract Subclass

Summary
Some features of a class are used only in some its instances

Goal
Extract a subclass for these features

Mechanics
- define a new subclass
- provide appropriate constructors for the subclass (use *Factory Method* if the subclass is to be hidden from clients)
- replace calls to superclass constructor with a call to the subclass one
- push down selected methods/fields to the subclass
- eliminate the fields that used to differentiate the behavior of original class which is now indicated by the inheritance
- compile & test

Inline Class

Name
Inline Class

Summary
A class does not earn for itself

Goal
Move its features to another class and delete it

Mechanics
- move the public protocol (with *Extract Interface*?) from the inlined class to the absorbing one
- declare the public methods of the inlined class in the source class and delegate the methods there
- change all client references from the source class to the absorbing one
- compile & test
- move fields & methods to the absorbing class

Example: Inline Class

```
public class Driver {
    private DrivingLicense license;
    private String name;

    public String getName() {
        return name;
    }

    public DrivingLicense getLicense() {
        return license;
    }
}

public class DrivingLicense {
    private String number;

    public void setNumber(String number) {
        this.number = number;
    }

    public String getNumber() {
        return number;
    }
}
```

Example: Inline Class

```
public class Driver {
    private DrivingLicense license;
    private String name;

    public String getName() {
        return name;
    }

    public DrivingLicense getLicense() {
        return license;
    }

    public void setLicenseNumber(String number) {
        license.setNumber(number);
    }

    public String getLicenseNumber() {
        return license.getNumber();
    }
}
```

Example: Inline Class

```
public class Driver {
    private DrivingLicense license;
    private String name;
    private String licenseNumber;

    public String getName() {
        return name;
    }

    public DrivingLicense getLicense() {
        return license;
    }

    public void setLicenseNumber(String number) {
        licenseNumber = number;
    }

    public String getLicenseNumber() {
        return licenseNumber;
    }
}
```

Example: Inline Class

```
public class Driver {

    private String name;
    private String licenseNumber;

    public String getName() {
        return name;
    }

    public void setLicenseNumber(String number) {
        licenseNumber = number;
    }

    public String getLicenseNumber() {
        return licenseNumber;
    }
}

public class LicenseNumber {
    // empty - to be removed
}
```

Replace Method with Method Object

Name
Replace Method with Method Object

Summary
A long method uses local variables so that it cannot be split

Goal
Turn the method into its own object

Mechanics

- create a new class and name it appropriately
- give it a final field for the object that hosted original method and fields for temporary variables and parameters
- create a constructor that takes a source object and each parameter
- move the original method into the new class
- compile
- replace the call to the method with creation of an instance of the new class and call its method

Example: Replace Method with Method Object

```
public class Account {

    int gamma(int inputVal, int quantity, int yearToDate) {
        int value1 = (inputVal * quantity) + delta()
        int value2 = (inputVal * yearToDate) + 100;
        if ((yearToDate - value1) > 100) {
            value2 -= 20;
        }
        int value3 = value2 * 7;

        return value3 - 2 * value1;
    }
}
```

example by M. Fowler

Example: Replace Method with Method Object

```
public class Gamma {
    private final Account account;
    private int inputVal;
    private int quantity;
    private int yearToDate;
    private int value1;
    private int value2;
    private int value3;

    public Gamma(Account source, int inputVal, int quantity,
        int yearToDate) {
        //... copy arguments
    }

    int compute() {
        value1 = (inputVal * quantity) + delta()
        value2 = (inputVal * yearToDate) + 100;
        if ((yearToDate - value1) > 100) {
            value2 -= 20;
        }
        value3 = value2 * 7;

        return value3 - 2 * value1;
    }
}
```

example by M. Fowler

Example: Replace Method with Method Object

```
public class Account {

    int gamma(int inputVal, int quantity, int yearToDate) {
        Gamma gamma = new Gamma(this, inputVal, quantity,
            yearToDate);
        return gamma.compute();
    }
}
```

example by M. Fowler

Form Template Method

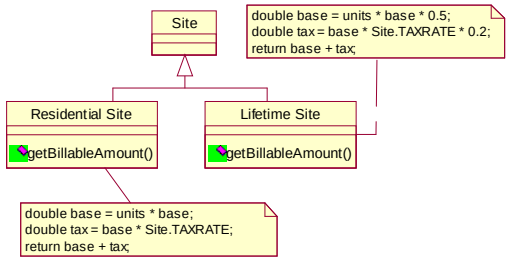
Name
Form Template Method

Summary
Two methods in subclasses perform similar steps in same order

Goal
Give them same signature and then pull them up

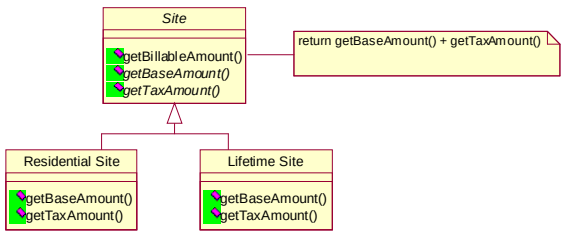
Mechanics
- decompose methods so that extracted methods are either identical or completely different (*Extract Method*, *Inline Method*)
- pull up the identical methods into the superclass
- rename different methods so the signatures of all methods at each step are the same, compile & test
- pull up one of original methods; make signatures of different methods abstract at superclass
- compile & test
- proceed with remaining methods, compile & test

Example: Form Template Method



example by M. Fowler

Example: Form Template Method



example by M. Fowler

Agenda

Refactorings related to
- incomplete library class
- inappropriate algorithm

Substitute Algorithm

Name
Substitute Algorithm

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

Goal
Replace the body of the method with the new algorithm

Mechanics
- prepare your alternative algorithm and get it compiling
- run the new algorithm against the tests
- if tests fail, use the old algorithm for comparison in testing and debugging

Example: Substitute Algorithm

```
String foundPerson(String[] people)  {

   for (int i = 0; i < people.length; i++)   {
      if (people[i].equals ("Don"))  {
         return "Don";
      }

      if (people[i].equals ("John"))  {
         return "John";
      }

      if (people[i].equals ("Kent"))  {
         return "Kent";
      }
   }

   return "";
}
```

example by M. Fowler

Example: Substitute Algorithm

```
String foundPerson(String[] people)  {

   List candidates = Arrays.asList(new String[] {
      "Don", "John", "Kent"});

   for (int i = 0; i < people.length; i++) {
      if (candidates.contains(people[i]))
          return people[i];
   }

   return "";
}
```

```
assertEquals("Don", foundPerson(new String[] {
    "Alice", "Don", "Mary"}));

assertEquals("", foundPerson(new String[] {
    "Alice", "Mike", "Mary"}));

assertNotEquals("", foundPerson(new String[] {}));
```

example by M. Fowler

Introduce Assertion

Name
Introduce Assertion

Summary
A section of code assumes sth about the state of program

Goal
Make the assumption explicit with an assertion

Mechanics
- add an assertion at condition assumed always to be true
- assertions are supposed not to change the behavior

Example: Introduce Assertion

```
public class Employee {
    private static final double NULL_EXPENSE = -1.0;
    private double _expenseLimit = NULL_EXPENSE;
    private Project _primaryProject;

    double getExpenseLimit() {
        return (_expenseLimit != NULL_EXPENSE) ?
            _expenseLimit :
            _primaryProject.getMemberExpenseLimit();
     }
}
```

```
public class Employee {
    private static final double NULL_EXPENSE = -1.0;
    private double _expenseLimit = NULL_EXPENSE;
    private Project _primaryProject;

    double getExpenseLimit() {
        assert(_expenseLimit != NULL_EXPENSE
            || _primaryProject != null);
        return (_expenseLimit != NULL_EXPENSE) ?
            _expenseLimit :
            _primaryProject.getMemberExpenseLimit();
     }
}
```

example by M. Fowler

Introduce Foreign Method

Name
Introduce Foreign Method

Summary
A server class needs a new method, but cannot be modified

Goal
Create class in the client class and pass a server class instance to it as the first argument

Mechanics
- create the needed method in the client class, passing all required data as parameters
- make an instance of the server class the first parameter
- comment appropriately the method as foreign to avoid accidental execution

Example: Introduce Foreign Method

```
Date nextDay = new Date(date.getYear(), date.getMonth(),
    date.getDate() + 1);
```

```
Date nextDay = nextDay(date);

private static Date nextDay(Date date) {
    return new Date(date.getYear(), date.getMonth(),
        date.getDate() + 1);
}
```

example by M. Fowler

Introduce Local Extension

Name
Introduce Local Extension

Summary
A server class needs a new method, but cannot be modified

Goal
Create a new class with extra method. Make it a wrapper or subclass of the original

Mechanics
- create an extension class as either wrapper or subclass of the original
- add converting constructors to the extension
- add new features to the extension
- replace the original with the extension where needed
- move any foreign methods defined for this class up to now onto the extension

Example 1: Introduce Local Extension

```
Date nextDay = new Date(date.getYear(), date.getMonth(),
    date.getDate() + 1);
```

```
public class ExtendedDate extend Date {
    public ExtendedDate(String str) {
        super(str);
    }

    public ExtendedDate(Date date) {
        super(date.getTime());
    }

    public Date nextDay() {
        return new Date(getYear(), getMonth(), getDate() + 1);
    }
}
```

example by M. Fowler

Example 2: Introduce Local Extension

```
Date nextDay = new Date(date.getYear(), date.getMonth(),
    date.getDate() + 1);
```

```
public class ExtendedDate {
    private Date date;

    public ExtendedDate(String str) {
        this.date = new Date(str);
    }

    public ExtendedDate(Date date) {
        this.date = date;
    }

    public Date nextDay() {
        return new Date(date.getYear(), date.getMonth(),
            date.getDate() + 1);
    }
}
```

example by M. Fowler

Replace Recursion with Iteration

Name
Replace Recursion with Iteration

Summary
Code that uses recursion is hard to understand

Goal
Replace recursion with iteration

Mechanics
- determine the base case of the recursion
- implement a loop that will iterate until the base case is reached
- make a progress towards the base case; send the new arguments to the top of the loop instead to the recursive method

Example: Replace Recursion with Iteration

```
public class CountDown {
   public void countDown(int n) {
      if (n == 0)
         return;
      System.out.println(n + "...");
      waitASecond();
      countDown(n - 1);
   }

   public void waitASecond() {
      try {
         Thread.sleep(1000);
      } catch (InterruptedException ignore) { }
   }

   public static void main(String[] args) {
      CountDown c = new CountDown();
      c.countDown(10);
   }
}
```

base case

progress

example by I. Mitrovic

Example: Replace Recursion with Iteration

```
public class CountDown {
   public void countDown(int n) {
      for (int i = n; i > 0; i--) {
         System.out.println(n + "...");
         waitASecond();
      }
   }

   public void waitASecond() {
      try {
         Thread.sleep(1000);
      } catch (InterruptedException ignore) { }
   }

   public static void main(String[] args) {
      CountDown c = new CountDown();
      c.countDown(10);
   }
}
```

example by I. Mitrovic

Replace Iteration with Recursion

Name
Replace Iteration with Recursion

Summary
It is not obvious what each iteration in loop is doing

Goal
Replace iteration with recursion

Mechanics
- identify the candidate loop that modifies one or more scoped locals and then returns a result based on their final values
- move the loop into a new function
- compile & test
- replace the loop with a function that accepts the local variables, and which returns the final result

Replace Iteration with Recursion

Name
Replace Iteration with Recursion

Mechanics
- *if* statement, tests the looping condition
- the *then* clause calculates/returns the final result
- the *else* clause makes the recursive call, with appropriately modified parameters
- compile & test

Example: Replace Iteration with Recursion

```
int greatestCommonDivisor (int a, int b) {
   while (a != b) {
      if (a > b) {
         a -= b;
      } else if (b > a) {
         b -= a;
      }
   }
}
```

base case

progress

```
int greatestCommonDivisor (int a, int b) {
   if (a != b) {
      if (a > b) {
         return greatestCommonDivisor(a - b, b);
      } else if (b > a) {
         return greatestCommonDivisor(a, b - a);
      }
   } else {
      return a;    // a == b
   }
}
```

example by I. Mitrovic

Q&A