<epam> Excellence in Software Engineering

BDO

JAVA ACADEMY

Refactoring

Fejér Attila

attila@javadev.hu

INTRODUCTION

Refactoring

- **Refactoring:** *a change made to the internal structure of software to make it easier to understand and cheaper to modify without changing its observable behavior.* (**Martin Fowler**: Refactoring Improving the Design of Existing Code)

Develompent and Refactoring

- When you add function, you shouldn't be changing existing code; you are just adding new capabilities.
- When you refactor, you make a point of not adding function; you only restructure the code.

- As you develop software you should swap these activities frequently.

Why should you refactor?

- **Improve the design of the software**
- **Make code easier to understand**
- **Helps to find bugs**
- **Program faster**
- **Eliminate code duplications**

- **When to refactor?**
  - First time just write the code
  - Second time you feel the smell
  - The third time you do something similar, you refactor

Down side of refactoring

- **Sometimes refactoring is difficult**
  - Database model
  - Published interfaces
  - Design changes
    - Security, architecture
- **Don’t refactor**
  - Code does not work
  - Close to deadline

Refactoring and performance

- Refactoring makes the software more amenable to performance tuning
- Refactoring often makes changes that will cause the program to run more slowly
  - Another level of indirection
- If you optimize all the code equally, you end up with 90 percent of the optimizations wasted
- Tool: **Profiler**

BAD SMELLS IN CODE

Bad smells in code

- Duplicate Code
- Long Method
- Large Class
- Long Parameter List
- Divergent Change
- Shotgun Surgery
- Feature Envy
- Data Clumps

Bad smells in code

- Magic constants
- Primitive Obsession
- Switch Statements
- Parallel Inheritance Hierarchies
- Lazy Class
- Speculative Generality
- Temporary Field
- Message Chains (*train wreck*)

Bad smells in code

- Middle Man
- Inappropriate Intimacy
- Alternative Classes with Different Interfaces
- Incomplete Library Class
- Data Class
- Refused Bequest
- Comment

UNIT TESTS AND REFACTORING

Unit Tests

- Refactoring requires working automated unit tests
- The test is another client for the refactored class
  - Make sure to maintain it when you refactor
  - Sometimes they need special attention
- Changes in the production code will typically involve changes in the test code as well
- Run the tests after every refactoring step

Refactoring

REFACTORING CATALOGUE

Format of the refactorings

- Name
- Description
- Motivation
- Steps
- Examples

Refactoring catalogue

- Composing methods
- Moving Features Between Objects
- Organizing data
- Simplifying Conditional Expressions
- Making Method Calls Simpler
- Dealing with Generalization

Refactoring

COMPOSING METHODS

Composing methods

- Extract Method
- Inline Method
- Inline Temp
- Replace Temp with Query
- Introduce Explaining Variable
- Split Temporary Variable
- Remove Assignments to Parameters
- Replace Method with Method Object
- Substitute Algorithm

Extract Method

- **Description**
  - Turn a fragment into a method whose name explains the purpose of it.
- **Motivation**
  - Short, well-named methods
- **Cases:**
  - No Local Variables
  - Using Local Variables
  - Reassigning a Local Variable

Extract Method example

```
void printAccount(double amount) {
   printBanner();

   //print details
   System.out.println ("name: " + this.name);
   System.out.println ("amount: " + amount);
}
```

```
void printAccount(double amount) {
   printBanner();
   printDetails(amount);
}

void printDetails (double amount) {
   System.out.println ("name: " + this.name);
   System.out.println ("amount: " + amount);
}
```

Inline Method

- **Description**
  - Put the method's body into the body of its callers and remove the method
- **Motivation**
  - Body is as clear as the name
  - Needless indirection is irritating
- **Note**
  - Don't inline if subclasses override the method

Inline Method example

```
int getRating() {
   return (moreThanFiveLateDeliveries()) ? 2 : 1;
}

boolean moreThanFiveLateDeliveries() {
   return this.numberOfLateDeliveries > 5;
}
```

```
int getRating() {
  return (this.numberOfLateDeliveries > 5) ? 2 : 1;
}
```

Inline Temp

- **Description**
  - Replace all references to that temp with the expression
- **Motivation**
  - The temp is getting in the way of other refactorings
- **Note**
  - Check the temp is really assigned only once
    - final keyword

Inline Temp example

```
double basePrice =
  anOrder.basePrice();

return (basePrice > 1000);
```

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

Replace Temp with Query

- **Description**
  - Extract the expression into a method
  - Replace all references to the temp with the expression
  - The new method can be used in other methods
- **Motivation**
  - Temps tend to encourage longer methods
  - The temp is getting in the way of other refactorings
- **Note**
  - Check the temp is really assigned only once
    - **final** keyword
  - Ensure the extracted method is free of side effects

Replace Temp with Query

```
double basePrice = _quantity * _itemPrice;
if (basePrice > 1000)
   return basePrice * 0.95;
else
   return basePrice * 0.98;
```

```
if (basePrice() > 1000)
   return basePrice() * 0.95;
else
   return basePrice() * 0.98;

[...]
double basePrice() {
   return _quantity * _itemPrice;
}
```

Introduce Explaining Variable

- **Description**
  - Put the result of the expression, or parts of the expression, in a temporary variable with a name
- **Motivation**
  - Expressions can become very complex and hard to read
- **Note**
  - Declare a final temporary variable
  - Use meaningful names

Introduce Explaining Variable

```
if ((platform.toUpperCase().indexOf("MAC") > -1) &&
   (browser.toUpperCase().indexOf("IE") > -1) &&

   wasInitialized() && resize > 0)

   [...]
```

```
final boolean isMacOs     = platform.toUpperCase().indexOf("MAC") > -1;
final boolean isIEBrowser = browser.toUpperCase().indexOf("IE")  > -1;
final boolean wasResized  = resize > 0;

if (isMacOs && isIEBrowser && wasInitialized() && wasResized)
[...]
```

Split Temporary Variable

- **Description**
  - Make a separate temporary variable for each assignment
- **Motivation**
  - Variable with more than one responsibility should be replaced with a temp for each responsibility
- **Note**
  - Don’t split collecting temporary variable

Split Temporary Variable

```
double temp = 2 * (this.height + this.width);
System.out.println (temp);
temp = this.height * this.width;
System.out.println (temp);
```

```
final double perimeter = 2 * (this.height + this.width);
System.out.println (perimeter);
final double area = this.height * this.width;
System.out.println (area);
```

Remove Assignments to Parameters

- **Description**
  - The code assigns to a parameter, use a temporary variable instead
- **Motivation**
  - Confusion between pass by value and pass by reference
  - The parameter should represent what has been passed in

Remove Assignments to Parameters

```
int discount (int inputVal, int quantity, int yearToDate) {

   if (inputVal > 50) inputVal -= 2;
   [...]
}
```

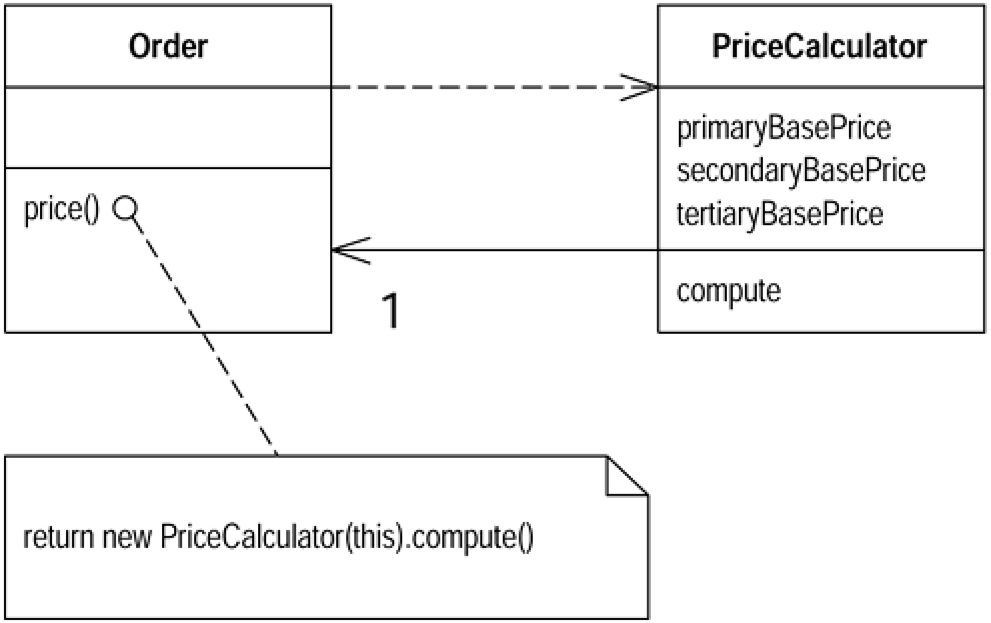
```
int discount (int inputVal, int quantity, int yearToDate) {
   int result = inputVal;
   if (inputVal > 50) result -= 2;
   [...]
}
```

Replace Method with Method Object

- **Description**
  - Turn the method into its own object so that all the local variables become fields on that object
  - You can then decompose the method into other methods on the same object
- **Motivation**
  - The difficulty in decomposing a method lies in local variables
  - Get rid of long parameter lists
  - Separate class responsible for the calculation

Replace Method with Method Object

```
class Order{
    double price() {
           double primaryBasePrice;
           double secondaryBasePrice;
           double tertiaryBasePrice;
       // calculation
    }
    [...]
}
```



Substitute Algorithm

- **Description**
  - Replace the body of the method with the new algorithm
- **Motivation**
  - Sometimes you just reach the point at which you have to remove the whole algorithm and replace it with something simpler

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 "";
}
```

```
String foundPerson(String[] people){
    List<String> 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 "";
}
```

Summarization

- Extract Method
- Inline Method
- Inline Temp
- Replace Temp with Query
- Introduce Explaining Variable
- Split Temporary Variable
- Remove Assignments to Parameters
- Replace Method with Method Object
- Substitute Algorithm

Exercises

- ComposingMethods project
  - exercise1
  - exercise2
  - exercise3
  - exercise4

1. Refactor
2. Undo the refactorings

Refactoring

MOVING FEATURES BETWEEN OBJECTS

Moving Features Between Objects

- Move Method
- Move Field
- Extract Class
- Inline Class
- Hide Delegate
- Remove Middle Man
- Introduce Foreign Method
- Introduce Local Extension

Move Method

- **Description**
  - Create a new method with a similar body in the class it uses most
  - Either turn the old method into a simple delegation, or remove it altogether
- **Motivation**
  - Classes are collaborating too much and are too highly coupled
  - A method seems to reference another object more than the object it lives on (especially after moving fields)

Move Method

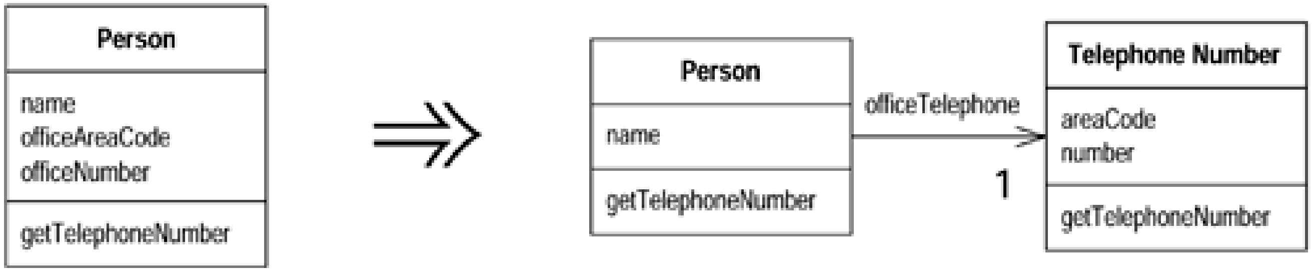
- **Note**
  - If the feature is used by other methods, consider moving them as well
  - Check the sub- and superclasses
  - You may need to create a new field in the source that can store the target
  - Consider leaving the source as delegating

Move Field

- **Description**
  - Create a new field in the target class, and change all its users
- **Motivation**
  - A design decision that is reasonable and correct one week can become incorrect in another
- **Note**
  - If the field is not private, check the subclasses
  - You may need to encapsulate the field

Extract Class

- **Description**
  - Create a new class and move the relevant fields and methods from the old class into the new class
- **Motivation**
  - The class became too complicated, too much responsibility
  - Data clumps



Extract Class

- **Note**
  - If the responsibilities of the old class no longer match its name, rename the old class
  - Make a link from the old to the new class
  - First move fields and then move the methods

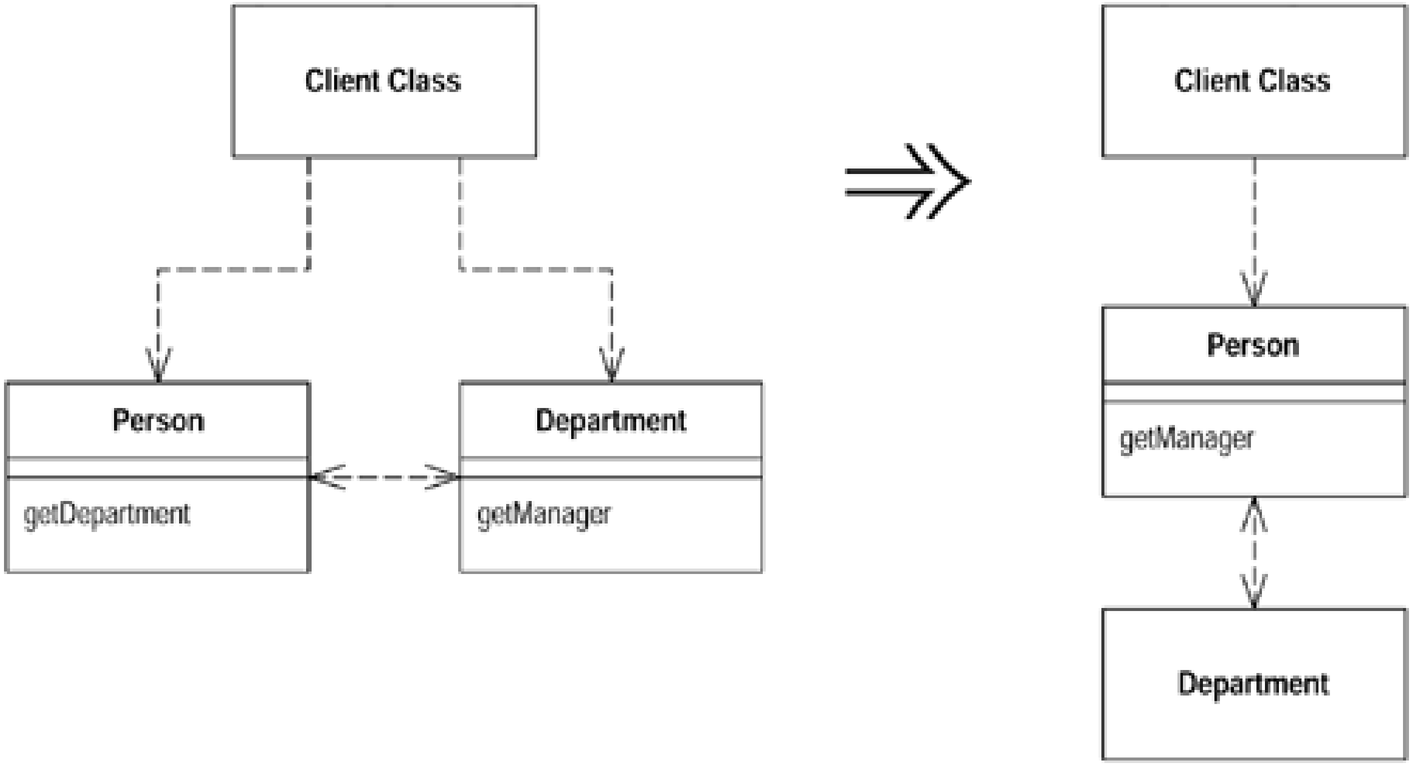
Inline Class

- **Description**
  - Move all its features into another class and delete it
- **Motivation**
  - Lazy class
- **Note**
  - You may consider creating a separate interface for the source class methods

Hide Delegate

- **Description**
  - Create methods on the server to hide the delegate
- **Motivation**
  - A client is calling a delegate class of an object
  - Remove dependency

Hide Delegate



Remove Middle Man

- **Description**
  - Get the client to call the delegate directly
- **Motivation**
  - Lots of delegations
  - Every new features means a new delegate
- **Note**
  - Generally this means a new dependency

Introduce Foreign Method

- **Description**
  - Create a method in the client class with an instance of the server class as its first argument
- **Motivation**
  - A server class you are using needs an additional method, but you can't modify the class
  - Avoid repetitive code
- **Note**
  - Just a work-around
  - Consider subclassing or using a wrapper

Introduce Foreign Method

```
Calendar calendar = Calendar.getInstance();
calendar.add(Calendar.DATE, 1);
return calendar.getTime();
```

```
Date newStart = nextDay(previousEnd);

public static Date nextDay(Date date) {
   Calendar calendar = Calendar.getInstance();
   calendar.setTime(date);
   calendar.add(Calendar.DATE, 1);
   return calendar.getTime();
}
```

Introduce Local Extension

- **Description**
  - Create a new class that contains these extra methods
  - Make this extension class a subclass or a wrapper of the original
- **Motivation**
  - There are more than one foreign methods
- **Note**
  - A subclass is easier to implement, but alters the object-creation

Summarization

- Move Method
- Move Field
- Extract Class
- Inline Class
- Hide Delegate
- Remove Middle Man
- Introduce Foreign Method
- Introduce Local Extension

Exercises

- MovingFeatures project
  - exercise1
  - exercise2

1. Refactor
2. Undo the refactorings

Refactoring

ORGANIZING DATA

Organizing Data

- Self Encapsulate Field
- Replace Data Value with Object
- Change Value to Reference
- Change Reference to Value
- Replace Array with Object
- Duplicate Observed Data
- Change Unidirectional Association to Bidirectional
- Change Bidirectional Association to Unidirectional

Organizing Data

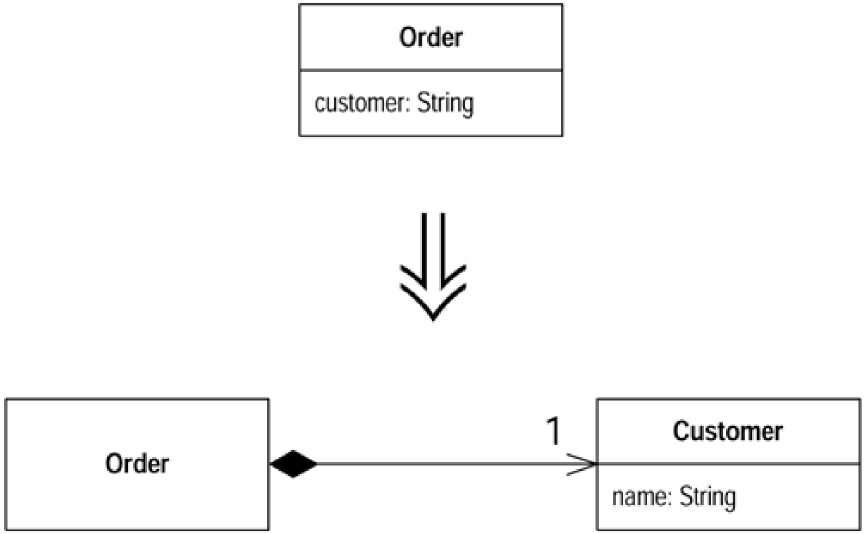
- Replace Magic Number with Symbolic Constant
- Encapsulate Field
- Encapsulate Collection
- Replace Type Code with Class
- Replace Type Code with Subclasses
- Replace Type Code with State/Strategy
- Replace Subclass with Fields

Self Encapsulate Field

- **Description**
  - Create getter and setter methods for the field and use only those to access the field from the same class
- **Motivation**
  - You want to override this variable access with a computed value in the subclass
- **Note**
  - Direct variable access is generally more simple

Replace Data Value with Object

- **Description**
  - Turn the data item into an object
- **Motivation**
  - You have a data item that needs additional data or behavior
  - Primitive obsession



Change Value to Reference

- **Description**
  - Turn the value object into a reference object
- **Motivation**
  - You have a class with many equal instances that you want to replace with a single object
- **Note**
  - Value objects are immutable
  - Each reference object stands for one object in the real world
  - Decide what object is responsible for providing access to the objects

Change Reference to Value

- **Description**
  - Turn reference object into a value object
- **Motivation**
  - You have a reference object that is small, immutable, and awkward to manage

Replace Array with Object

- **Description**
  - Replace the array with an object that has a field for each element
- **Motivation**
  - You have an array in which certain elements mean different things
- **Note**
  - Arrays should be used only to contain a collection of similar objects in some order

Replace Array with Object

```
String[] row = new String[3];
row[0] = "Liverpool";
row[1] = "15";
```

```
Performance row = new Performance();
row.setName("Liverpool");
row.setWins("15");
```

Duplicate Observed Data

- **Description**
  - Move or copy the data to a domain object
  - Set up a mechanism for synchronizing data
- **Motivation**
  - You have domain data available only in a GUI control
- **Note**
  - Separate user interface from business logic

Change Unidirectional Association to Bidirectional

- **Description**
  - Add pointers, and change modifiers to update both sets
- **Motivation**
  - You have two classes that need to use each other's features, but there is only a one-way link
- **Note**
  - Create tests for testing accessors

Change Unidirectional Association to Unidirectional

- **Description**
  - Drop the unneeded end of the association
- **Motivation**
  - You have a two-way association but one class no longer needs features from the other
- **Note**
  - Bidirectional accessors means complexity of maintaining the two-way links
  - Use bidirectional associations only when it’s necessary

Replace Magic Number with Symbolic Constant

- **Description**
  - Create a constant, name it after the meaning, and replace the number with it
- **Motivation**
  - You have a literal number with a particular meaning

```
double potentialEnergy(double mass, double height) {
   return mass * 9.81 * height;
}
```

```
static final double GRAVITATIONAL_CONSTANT = 9.81;
double potentialEnergy(double mass, double height) {
   return mass * GRAVITATIONAL_CONSTANT * height;
}
```

Encapsulate Field

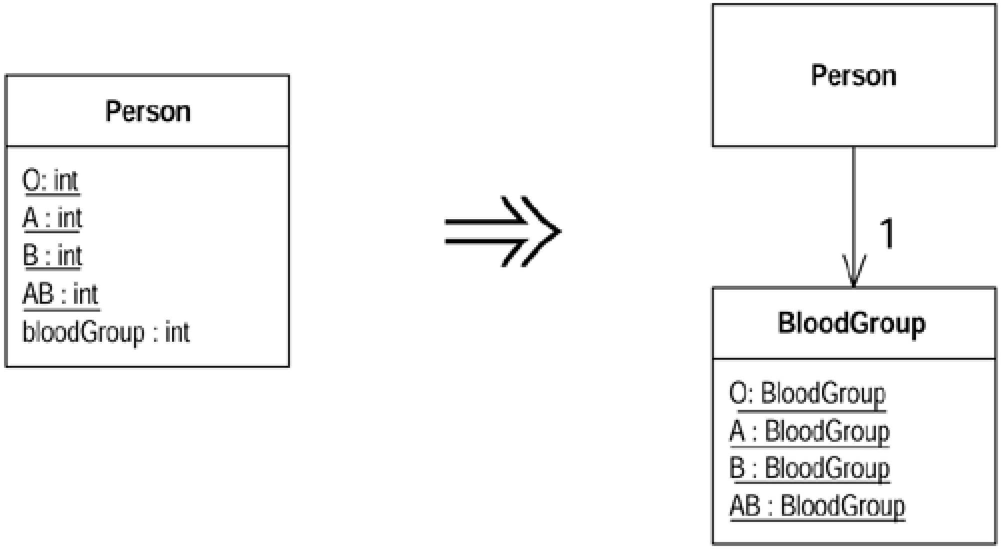
- **Description**
  - Make the field private and provide accessors
- **Motivation**
  - There is a public field
  - Encapsulation, data hiding
- **Note**
  - A client may alter a field by calling a modifier on the object

Encapsulate Collection

- **Description**
  - Make it return a read-only view and provide add/remove methods
- **Motivation**
  - A method returns a collection
- **Note**
  - A simple getter for a modifiable collection allows clients to manipulate the contents
  - There should not be a setter for collection
    - there should be operations to add and remove elements

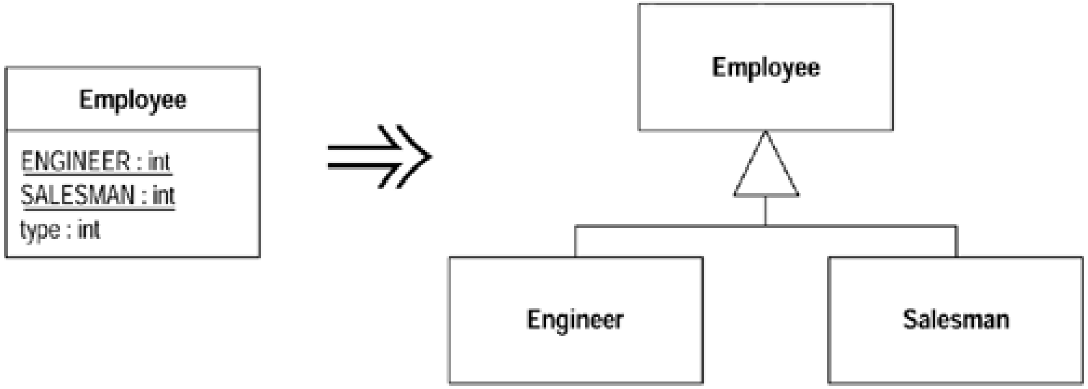
Replace Type Code with Type

- **Description**
  - Replace the type code with a new enum
- **Motivation**
  - A class has a numeric type code that does not affect its behavior



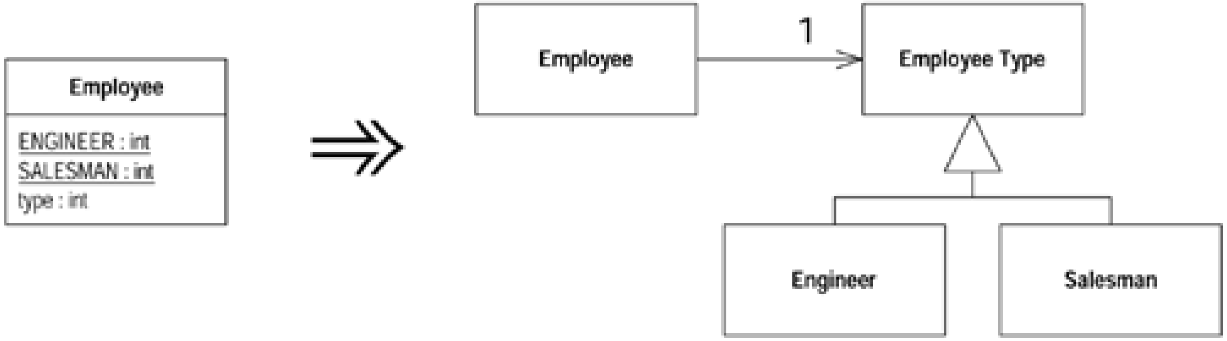
Replace Type Code with Subclasses

- **Description**
  - Replace the type code with a new subclass
- **Motivation**
  - You have an immutable type code that affects the behavior of a class
  - Avoid conditional statements



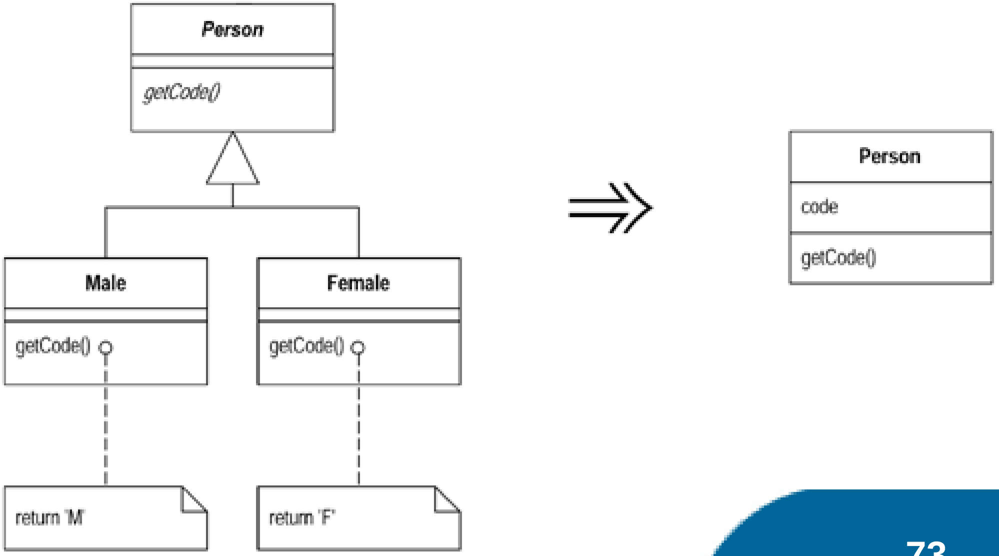
Replace Type Code with State/Strategy

- **Description**
  - Replace the type code with a state object
- **Motivation**
  - You have a type code that affects the behavior of a class, but you don't want to use subclassing
  - Type code can change dynamically



Replace Subclass with Fields

- **Description**
  - Change the methods to superclass fields and eliminate the subclasses
- **Motivation**
  - Subclasses should add features or allow the behavior to vary
  - Constant methods



Summarization

- Self Encapsulate Field
- Replace Data Value with Object
- Change Value to Reference
- Change Reference to Value
- Replace Array with Object
- Duplicate Observed Data
- Change Unidirectional Association to Bidirectional
- Change Bidirectional Association to Unidirectional

Summarization

- Replace Magic Number with Symbolic Constant
- Encapsulate Field
- Encapsulate Collection
- Replace Type Code with Class
- Replace Type Code with Subclasses
- Replace Type Code with State/Strategy
- Replace Subclass with Fields

Exercises

- Organizing data
  - exercise1
  - exercise2
- Refactor

SIMPLIFYING CONDITIONAL EXPRESSIONS

Simplifying Conditional Expressions

- Decompose Conditional
- Consolidate Conditional Expression
- Consolidate Duplicate Conditional Expression
- Remove Control Flag
- Replace Nested Conditional with Guard Clauses
- Replace Conditional with Polymorphism
- Introduce Null Object

Decompose conditional

- **Description**
  - Extract methods from the condition, then part, and else parts.
- **Motivation**
  - You have a complicated conditional (if-then-else) statement

Decompose conditional

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

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

Consolidate Conditional Expression

- **Description**
  - Consolidate conditional tests with same result into a single conditional expression and extract it
- **Motivation**
  - You have a sequence of conditional tests with the same result
  - Avoid repetitions
- **Note**
  - If the checks are independent don’t do the refactoring

Consolidate Conditional Expression

```
if (this.seniority < 2) return 0;
if (this.monthsDisabled > 12) return 0;
if (this.isPartTime) return 0;
```

```
if (isNotEligableForDisability()) return 0;
```

Consolidate Duplicate Conditional Fragments

- **Description**
  - Move the repeating fragment of code outside of the expression
- **Motivation**
  - The same fragment of code is in all branches of a conditional expression

Consolidate Duplicate Conditional Fragments

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

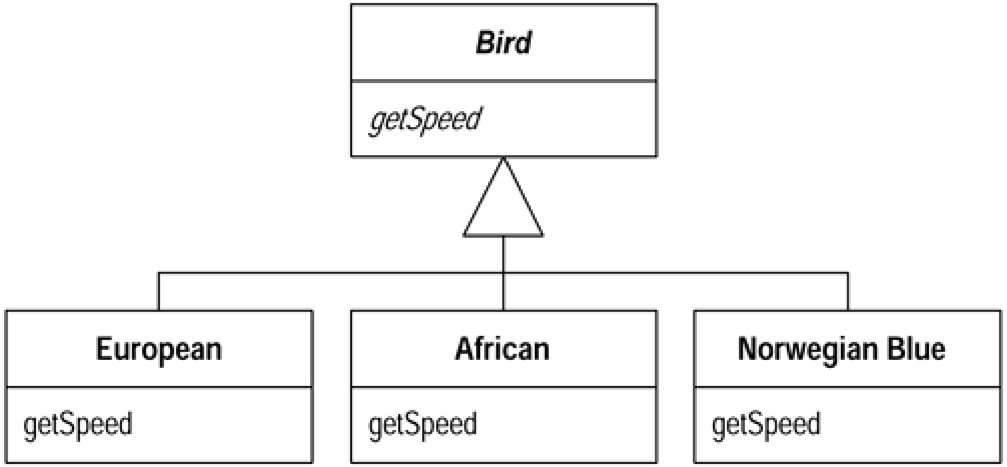
```
if (isSpecialDeal())
   total = price * 0.95;
else
   total = price * 0.98;
send();
```

Replace Conditional with Polymorphism

- **Description**
  - Move each leg of the conditional to an overriding method in a subclass. Make the original method abstract
- **Motivation**
  - You have a conditional that chooses different behavior depending on the type of an object
  - Avoid the instanceof keyword
- **Note**
  - You may need to change some private members to protected visibility

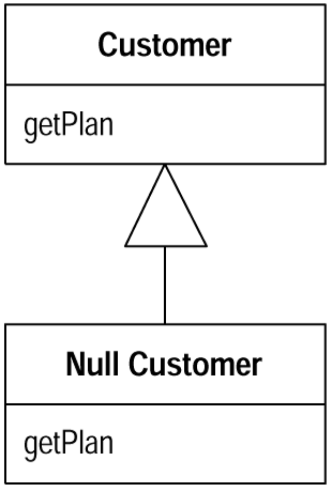
Replace Conditional with Polymorphism

```
double getSpeed() {
    switch (this.type) {
          case EUROPEAN: return getBaseSpeed();
          case AFRICAN: return getBaseSpeed() - getLoadFactor() *
    this.numberOfCoconuts;
          case NORWEGIAN_BLUE: return (this.isNailed) ? 0 :
    getBaseSpeed(this.voltage);
    }
    throw new RuntimeException ("Should be unreachable");
}
```



Introduce Null Object

- **Description**
  - Replace the null value with a null object
- **Motivation**
  - You have repeated checks for a null value
- **Note**
  - The null objects can be shared



Summarization

- Decompose Conditional
- Consolidate Conditional Expression
- Consolidate Duplicate Conditional Expression
- Remove Control Flag
- Replace Nested Conditional with Guard Clauses
- Replace Conditional with Polymorphism
- Introduce Null Object

Exercise

- **SimplifyingConditionals project**
  - exercise1

MAKING METHOD CALLS SIMPLER

Making Method Calls Simpler

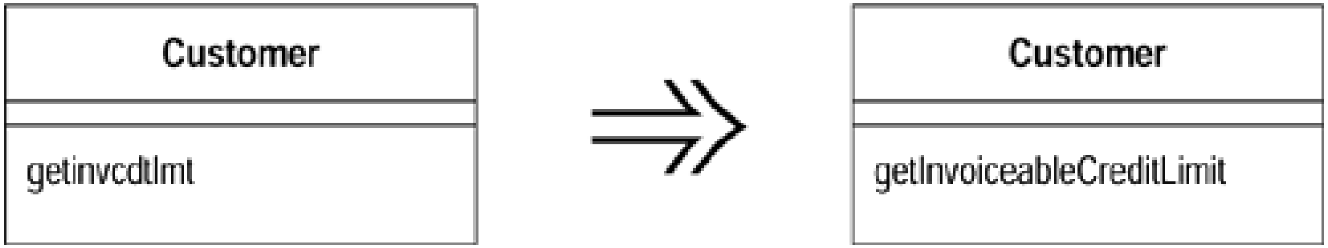
- Rename method
- Add Parameter
- Remove Parameter
- Separate Query from Modifier
- Parameterize Method
- Replace Parameter with Explicit Methods
- Preserve Whole Object
- Replace Parameter with Method

Making Method Calls Simpler

- Introduce Parameter Object
- Remove Setting Method
- Hide Method
- Replace Constructor with Factory Method
- Encapsulate Downcast
- Replace Error Code with Exception
- Replace Exception with Test

Rename Method

- **Description**
  - Change the name of the method
- **Motivation**
  - The name of a method does not reveal its purpose.

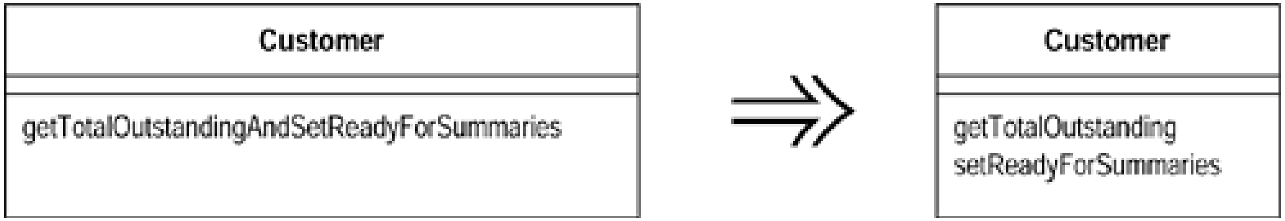


Add/Remove Parameter

- **Description**
  - Add/Remove a parameter for/from an object that can pass on this information
- **Motivation**
  - A method needs more/less information from its caller
- **Note**
  - Be careful with long parameter list
  - Consider alternatives

Separate Query from Modifier

- **Description**
  - Create two methods, one for the query and one for the modification
- **Motivation**
  - Make a difference between methods with side effects and those without
  - Rule of thumb: any method that returns a value should not have observable side effects



Customer

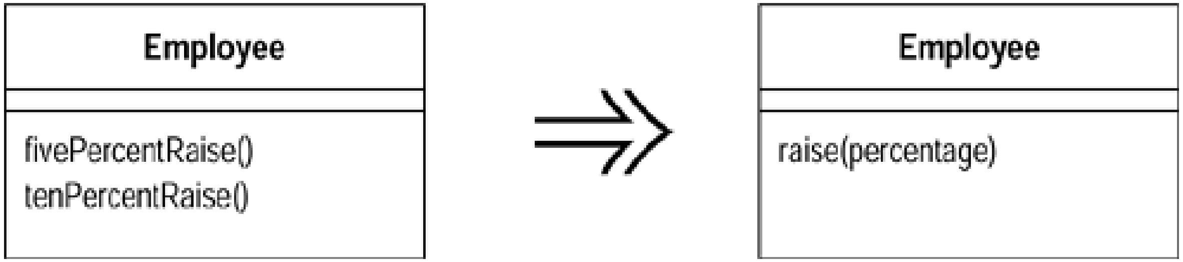
getTotalOutstandingAndSetReadyForSummaries

Customer

getTotalOutstanding
setReadyForSummaries

Parameterize Method

- **Description**
  - Create one method that uses a parameter for the different values
- **Motivation**
  - Several methods do similar things but with different values contained in the method body.



Replace Parameter with Explicit Methods

- **Description**
  - Create a separate method for each value of the parameter
- **Motivation**
  - You have a method that runs different code depending on the values of an enumerated parameter
- **Note**
  - Don’t use when the parameter values are likely to change a lot

Replace Parameter with Explicit Methods

```
void setValue(String name, int value) {
   if (name.equals("height"))
        this.height = value;
   if (name.equals("width"))
        this.width = value;
}
```

```
void setHeight(int arg) {
   this.height = arg;
}

void setWidth(int arg) {
   this.width = arg;
}
```

Preserve Whole Object

- **Description**
  - Send the whole object as parameter
- **Motivation**
  - You are getting several values from an object and passing these values as parameters in a method call
  - Shorter parameter list
- **Note**
  - Might introduce dependency

Preserve Whole Object

```
int low = daysTempRange().getLow();
int high = daysTempRange().getHigh();
withinPlan = plan.withinRange(low, high);
```

```
withinPlan = plan.withinRange(daysTempRange());
```

Replace Parameter with Method

- **Description**
  - Remove the parameter and let the receiver invoke the method
- **Motivation**
  - An object invokes a method, then passes the result as a parameter for a method. The receiver can also invoke this method.
  - Shorter parameter list
- **Note**
  - Might introduce additional dependency

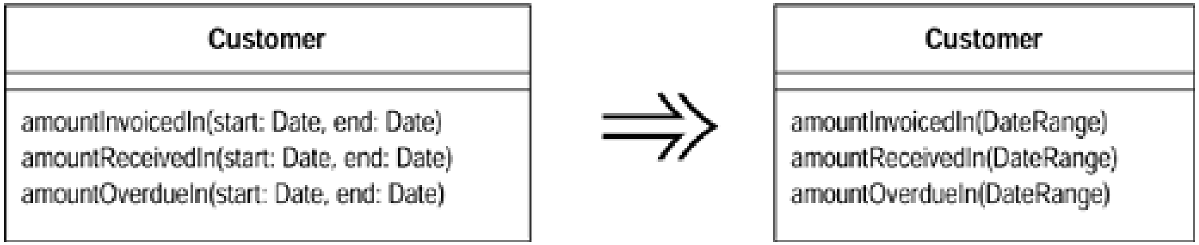
Replace Parameter with Method

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

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

Introduce Parameter Object

- **Description**
  - Replace some parameters with an object
- **Motivation**
  - You have a group of parameters that naturally go together
  - Data clumps



Customer

amountInvoicedIn(start: Date, end: Date)
amountReceivedIn(start: Date, end: Date)
amountOverdueIn(start: Date, end: Date)

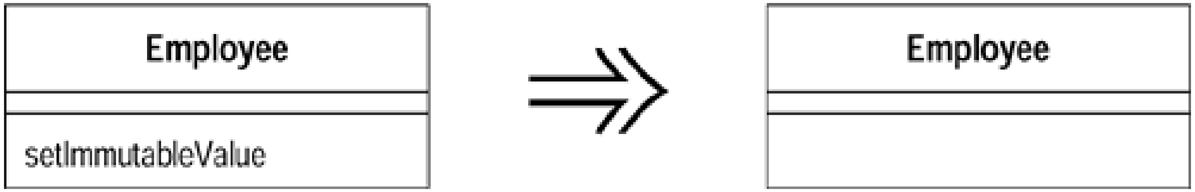
⇛

Customer

amountInvoicedIn(DateRange)
amountReceivedIn(DateRange)
amountOverdueIn(DateRange)

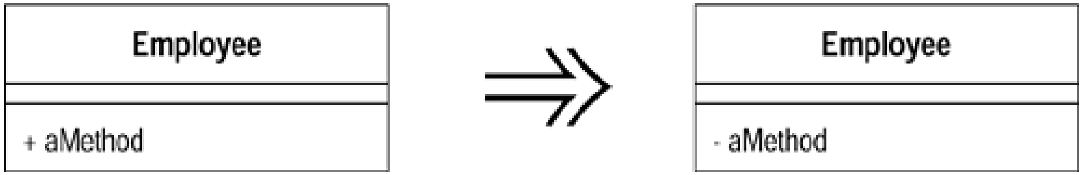
Remove Setting Method

- **Description**
  - Remove any setting method (or make them private) for that field
  - Optionally make the field final
- **Motivation**
  - A field should be set at creation time and never altered
  - Create immutable class



Hide Method

- **Description**
  - Make the method private
- **Motivation**
  - A method is not used by any other class
- **Hint**
  - Check regularly for unused methods



Replace Constructor with Static Factory Method

- **Description**
  - Replace the constructor with a static factory method
- **Motivation**
  - You want to do more than simple construction when you create an object
  - Factory can choose from many subclasses
  - Controlling access to limited resources
  - Different names for readablilty

Encapsulate Downcast

- **Description**
  - Move the downcast to within the method
- **Motivation**
  - A method returns an object that needs to be downcasted by its callers
- **Note**
  - Use template parameters if possible

```
Object lastReading() {
    return readings.lastElement();
}
```

```
Reading lastReading() {
    return (Reading)
           readings.lastElement();
}
```

Replace Error Code with Exception

- **Description**
  - Throw an exception instead of a returned error code or null value
- **Motivation**
  - A method returns a special code to indicate an error.
  - A method returns null to indicate an error.

```
int withdraw(int amount) {
   if (amount > _balance) return -1;
   else {
        this.balance -= amount;
        return 0;
   }
}
```

Replace Exception with Test

- **Description**
  - Change the caller to make the test first
- **Motivation**
  - Throwing a checked exception on a condition the caller could have checked first
  - Exceptions should be used for exceptional behavior

Replace Exception with Test

```
double getValueForPeriod (int periodNumber) {
   try {
        return this.values[periodNumber];
   } catch (ArrayIndexOutOfBoundsException e) {
        return 0;
   }
}
```

```
double getValueForPeriod (int periodNumber) {
   if (periodNumber >= this.values.length) return 0;
   return this.values[periodNumber];
}
```

Summarization

- Rename method
- Add Parameter
- Remove Parameter
- Separate Query from Modifier
- Parameterize Method
- Replace Parameter with Explicit Methods
- Preserve Whole Object
- Replace Parameter with Method

Summarization

- Introduce Parameter Object
- Remove Setting Method
- Hide Method
- Replace Constructor with Factory Method
- Encapsulate Downcast
- Replace Error Code with Exception
- Replace Exception with Test

Summarization

- MethodCalls
  - exercise1
  - exercise2

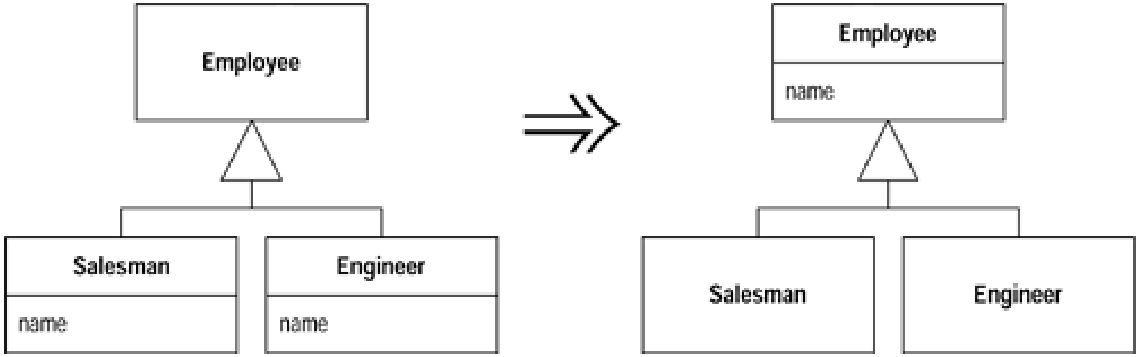
DEALING WITH GENERALIZATION

Dealing with Generalization

- Pull Up / Push Down Field
- Pull Up / Push Down Method
- Pull Up Constructor Body
- Extract Subclass
- Extract Superclass
- Extract Interface
- Collapse Hierarchy
- Replace Inheritance with Delegation
- Replace Delegation with Inheritance

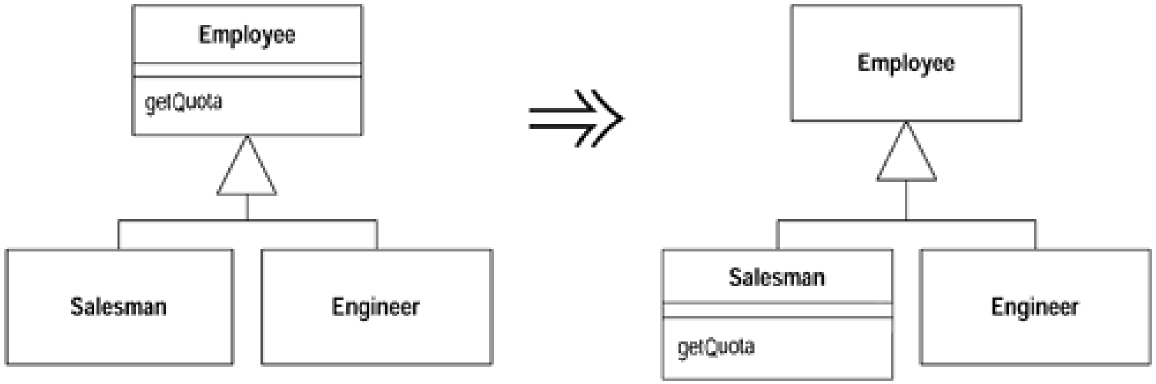
Pull Up Field

- **Description**
  - Move the field to the superclass
- **Motivation**
  - All subclasses have the same field



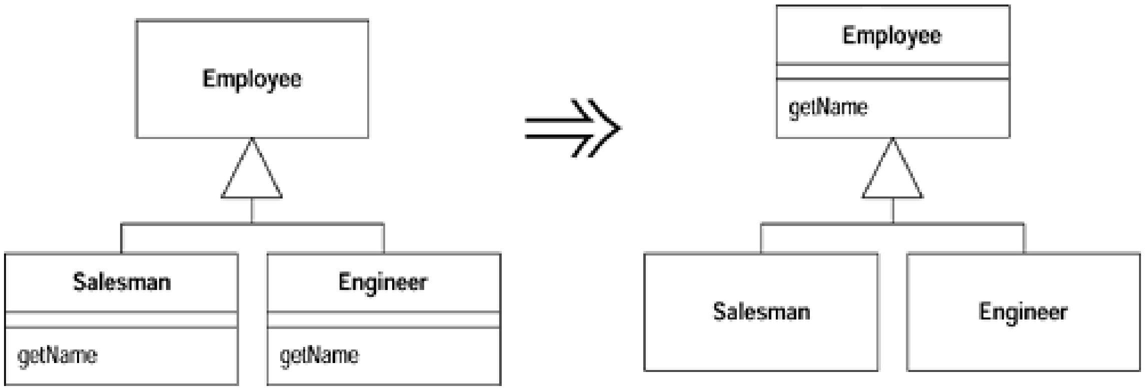
Push Down Field

- **Description**
  - Move the field to the subclasses.
- **Motivation**
  - A field is used only by some subclasses.



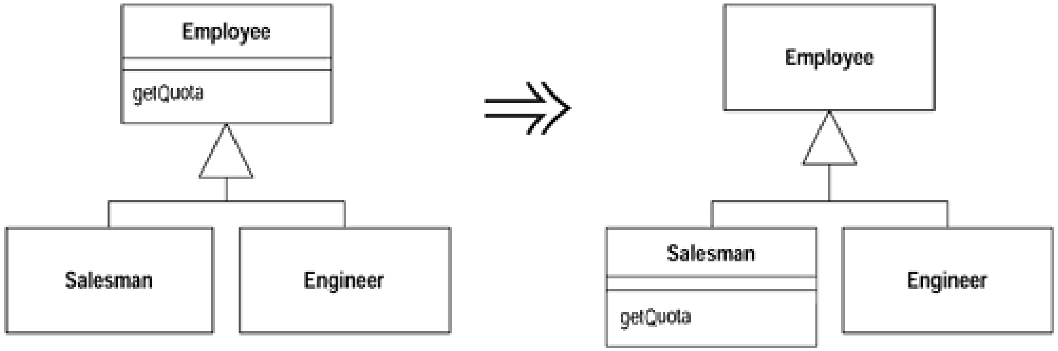
Pull Up Method

- **Description**
  - Move methods with identical result to superclass
- **Motivation**
  - Methods with identical results on subclasses



Push Down Method

- **Description**
  - Move behavior to the subclasses
- **Motivation**
  - Behavior on a superclass is relevant only for some of its subclasses

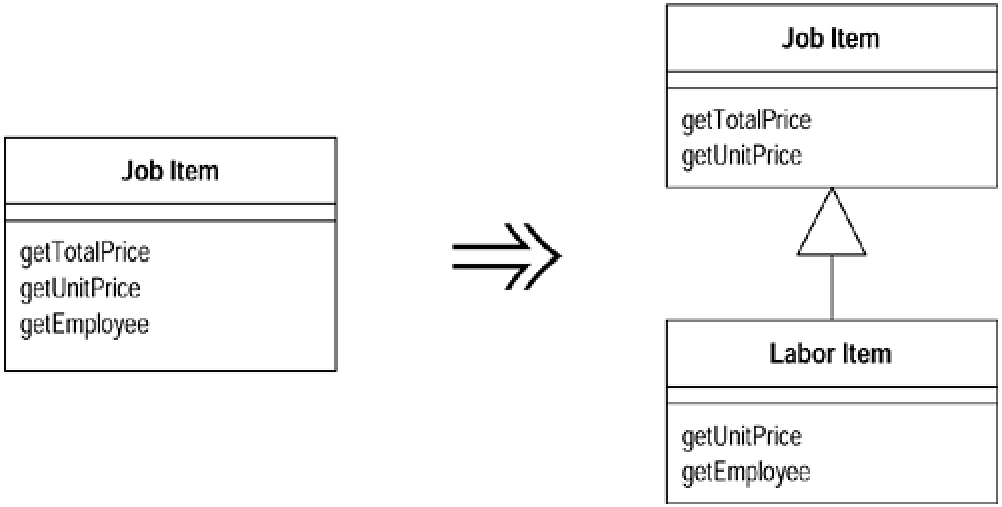


Pull Up Constructor Body

- **Description**
  - Create a superclass constructor
  - Call this from the subclass constructors
- **Motivation**
  - You have constructors on subclasses with mostly identical bodies.
  - Avoid repetition

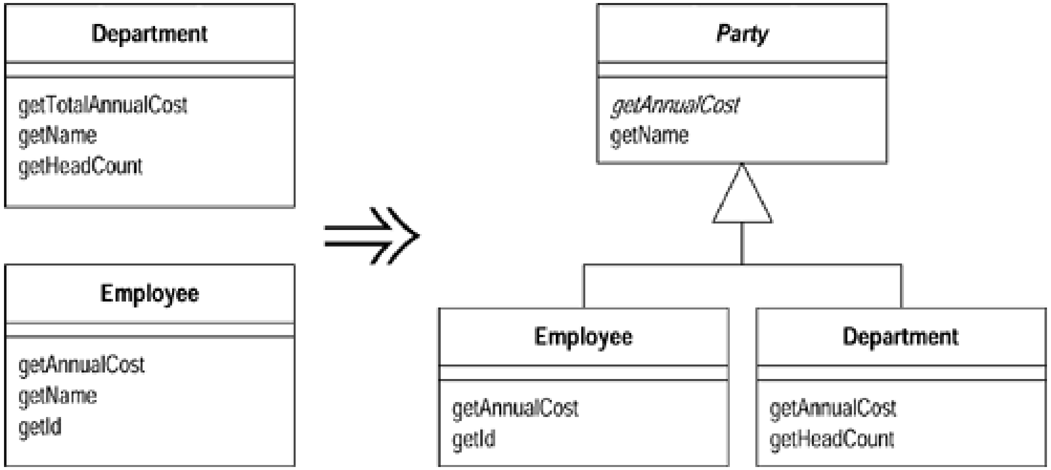
Extract Subclass

- **Description**
  - A class has features that are used only in some instances
- **Motivation**
  - Create a subclass for that subset of features.



Extract Superclass

- **Description**
  - Create a superclass and move the common features to the superclass.
- **Motivation**
  - You have two classes with similar features.

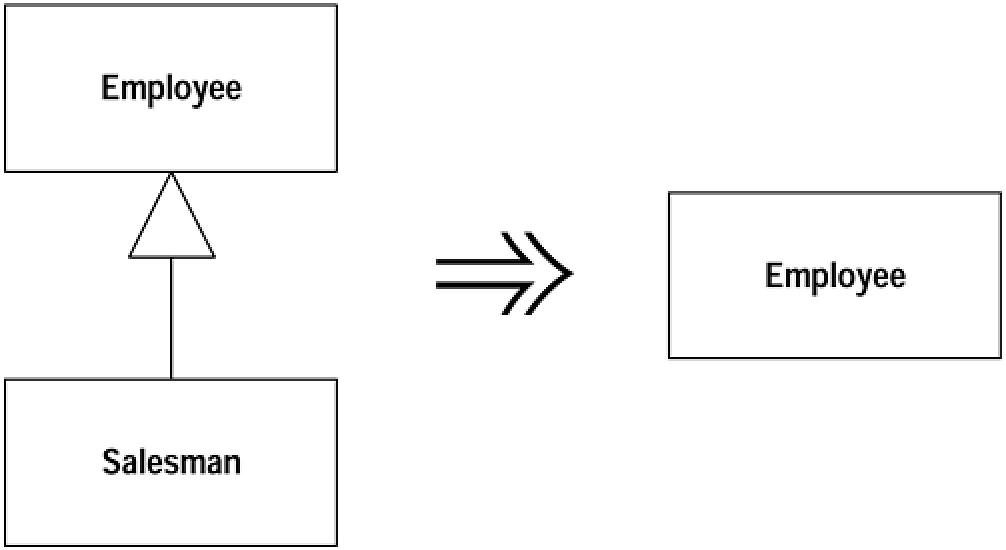


Extract Interface

- **Description**
  - Extract the subset of methods into an interface
- **Motivation**
  - Several clients use the same subset of a class's interface
  - Two classes have part of their interfaces in common
- **Hint**
  - Use extract interface for each role of a class

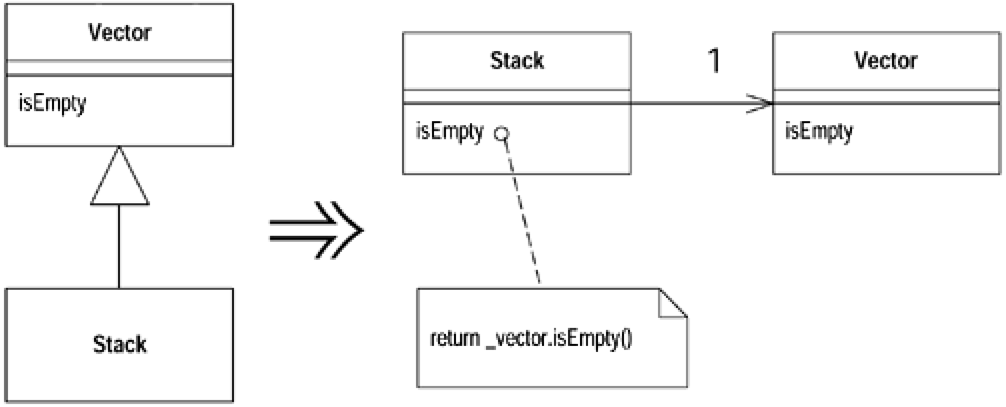
Collapse Hierarchy

- **Description**
  - Merge classes together
- **Motivation**
  - A superclass and subclass are not very different.



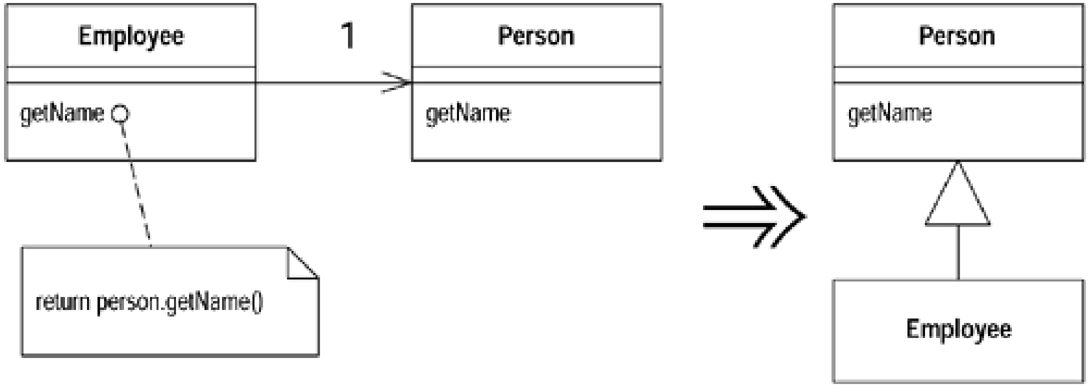
Replace Inheritance with Delegation

- **Description**
  - Create a field for the superclass
  - Methods delegate to the superclass
  - Remove the subclassing
- **Motivation**
  - A subclass uses only part of a superclasses interface or does not want to inherit data.



Replace Delegation with Inheritance

- **Description**
  - Make the delegating class a subclass of the delegate.
- **Motivation**
  - You're using delegation and are often writing many simple delegations for the entire interface.



Summarization

- Pull Up / Push Down Field
- Pull Up / Push Down Method
- Pull Up Constructor Body
- Extract Subclass
- Extract Superclass
- Extract Interface
- Collapse Hierarchy
- Replace Inheritance with Delegation
- Replace Delegation with Inheritance

Exercises

- Generalization
  - exercise1
  - exercise2

Refactoring

REFACTORING IN ECLIPSE

Key combinations

- Show Key Assist: CTRL + SHIFT + L
- Organize imports: CTRL + SHIFT + O
- Format code: CTRL + SHIFT + F
- Open type: CTRL + SHIFT + T
- Type hierarchy: CTRL + T
- Go to: CTRL + CLICK
- References: CTRL + SHIFT + G
- Refactorings: ALT + SHIT + T
- Source: ALT + SHIFT + S

Rename

- Alt + Shift + R
- Renames the selected element and (if enabled) corrects all references to the elements (also in other files).
- **Available:** Methods, method parameters, fields, local variables, types, type parameters, enum constants, compilation units, packages, source folders, projects

Move

- Alt + Shift + V
- Moves the selected elements and (if enabled) corrects all references to the elements (also in other files).
- **Available:** Instance method, one or more static methods, static fields, types, compilation units, packages, source folders and projects

Change Method Signature

- Alt + Shift + C
- Changes parameter names, parameter types, parameter order and updates all references to the corresponding method. Additionally, parameters and thrown exceptions can be removed or added and method return type and method visibility can be changed.
- **Available:** Methods

Extract Method

- Alt + Shift + M
- Creates a new method containing the statements or expression currently selected and replaces the selection with a reference to the new method. This feature is useful for cleaning up lengthy, cluttered, or overly-complicated methods.
- **Available:** Code fragment

Extract Local Variable

- Alt + Shift + L
- Creates a new variable assigned to the expression currently selected and replaces the selection with a reference to the new variable.
- **Available:** Local variables

Extract Constant

- Creates a static final field from the selected expression and substitutes a field reference, and optionally rewrites other places where the same expression occurs.
- **Available:** Constant expressions

Inline

- Alt + Shift + I
- Inline local variables, methods or constants.
- **Available:** Methods, static final fields or local variables

Convert Anonymous Class to Nested

- Converts an anonymous inner class to a member class.
- **Available:** Anonymous inner classes

Move Type to New File

- Creates a new Java compilation unit for the selected member type. For non-static member types, a field is added to allow access to the former enclosing instance, if necessary.
- **Available:** Member types

Convert Local Variable to Field

- Turn a local variable into a field. If the variable is initialized on creation, then the operation moves the initialization to the new field's declaration or to the class's constructors.
- **Available:** Local variables

Extract Superclass

- Extracts a common superclass from a set of sibling types. The selected sibling types become direct subclasses of the extracted superclass after applying the refactoring.
- **Available:** Types

Extract Interface

- Creates a new interface with a set of methods and makes the selected class implement the interface.
- **Available:** Types

Use Supertype Where Possible

- Replaces occurrences of a type with one of its supertypes after identifying all places where this replacement is possible.
- **Available:** Types

Push Down / Pull Up

- Moves a set of methods and fields from a class to its superclass / subclasses.
- **Available:** One or more methods and fields declared in the same type

Extract Class

- Replaces a set of fields with new container object. All references to the fields are updated to access the new container object.
- **Available:** A number of fields or a type containing fields

Introduce Parameter Object

- Replaces a set of parameters with a new class, and updates all callers of the method to pass an instance of the new class as the value to the introduce parameter.
- **Available:** Methods

Introduce Factory

- Creates a new factory method, which will call a selected constructor and return the created object. All references to the constructor will be replaced by calls to the new factory method.
- **Available:** Constructor declarations

Encapsulate Field

- Replaces all references to a field with getter and setter methods.
- **Available:** Field

Excellence in
Software Engineering

THANK YOU FOR YOUR KIND ATTENTION!