



Refactoring Code Smells



If it Stinks, change it!



What is Refactoring?

What is Refactoring?

- ☑ A series of **small** steps, each of which changes the program's **internal structure** without changing its **external behavior** - Martin Fowler

What is Refactoring?

- ☑ A series of **small** steps, each of which changes the program's **internal structure** without changing its **external behavior** - Martin Fowler
- ☑ Verify no change in external behavior by
 - ☑ Testing
 - ☑ Using the right tool - IDE
 - ☑ Formal code analysis by tool
 - ☑ Being very, very careful



What if you hear...




What if you hear...

- ☒ We'll just refactor the code to support logging

What if you hear...

- 
- ☒ We'll just refactor the code to support logging

What if you hear...

- 
- ☒ We'll just refactor the code to support logging
 - ☒ Can you refactor the code so that it authenticates against LDAP instead of Database?




What if you hear...

- ☒ We'll just refactor the code to support logging
- ☒ Can you refactor the code so that it authenticates against LDAP instead of Database?




What if you hear...

- ☒ We'll just refactor the code to support logging
- ☒ Can you refactor the code so that it authenticates against LDAP instead of Database?
- ☒ We have too much duplicate code, we need to refactor the code to eliminate duplication





What if you hear...

- ☒ We'll just refactor the code to support logging 
- ☒ Can you refactor the code so that it authenticates against LDAP instead of Database? 
- ☒ We have too much duplicate code, we need to refactor the code to eliminate duplication 






What if you hear...

- ☒ We'll just refactor the code to support logging 
- ☒ Can you refactor the code so that it authenticates against LDAP instead of Database? 
- ☒ We have too much duplicate code, we need to refactor the code to eliminate duplication 
- ☒ This class is too big, we need to refactor it






What if you hear...

- ☒ We'll just refactor the code to support logging 
- ☒ Can you refactor the code so that it authenticates against LDAP instead of Database? 
- ☒ We have too much duplicate code, we need to refactor the code to eliminate duplication 
- ☒ This class is too big, we need to refactor it 

What if you hear...

- ☒ We'll just refactor the code to support logging 
- ☒ Can you refactor the code so that it authenticates against LDAP instead of Database? 
- ☒ We have too much duplicate code, we need to refactor the code to eliminate duplication 
- ☒ This class is too big, we need to refactor it 
- ☒ Caching? 

What if you hear...

- ☒ We'll just refactor the code to support logging 
- ☒ Can you refactor the code so that it authenticates against LDAP instead of Database? 
- ☒ We have too much duplicate code, we need to refactor the code to eliminate duplication 
- ☒ This class is too big, we need to refactor it 
- ☒ Caching? 



Why do we Refactor?

Why do we Refactor?

- ☑ Helps us deliver **more business value faster**
- ☑ Improves the **design** of our software
 - ☑ Combat's "bit rot"
 - ☑ Easier to maintain and understand
 - ☑ Easier to facilitate change
 - ☑ More flexibility
 - ☑ Increased re-usability



Why do we Refactor?...

Why do we Refactor?...

- ☑ Minimizes *technical debt*
- ☑ Keep **development** at *speed*
- ☑ To make the software easier to **understand**
 - ☑ Write for people, not the compiler
 - ☑ Understand unfamiliar code
- ☑ To help find **bugs**
 - ☑ refactor while debugging to clarify the code

Readability

Which code segment is easier to read?

Sample 1

```
if (date.Before(Summer_Start) || date.After(Summer_End)){  
    charge = quantity * winterRate + winterServiceCharge;  
else charge = quantity * summerRate;  
  
}
```

Sample 2

```
if (IsSummer(date)) {  
    charge = SummerCharge(quantity);  
else charge = WinterCharge(quantity);  
  
}
```



When should you refactor?

When should you refactor?

- ☑ To add **new functionality**

- ☑ refactor existing code until you understand it
- ☑ refactor the design to make it simple to add

- ☑ To find **bugs**

- ☑ refactor to understand the code

- ☑ For **code reviews**

- ☑ immediate effect of code review
- ☑ allows for higher level suggestions

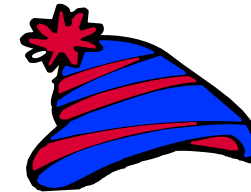
The Two Hats

Adding Function



- ☒ Add new capabilities to the system
- ☒ Adds new tests
- ☒ Get the test working

Refactoring



- ☒ Does not add any new features
- ☒ Does not add tests (but may change some)
- ☒ Restructure the code to remove redundancy

How do we Refactor?

- ☑ We look for Code-Smells
- ☑ Things that we suspect are not quite right or will cause us severe pain if we do not fix



2 Piece of Advice before Refactoring

2 Piece of Advice before Refactoring

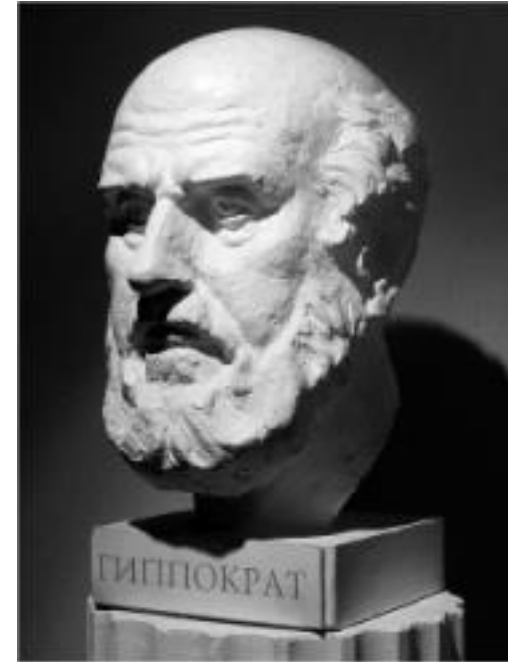


Baby Steps

2 Piece of Advice before Refactoring



Baby Steps



The Hippocratic Oath

First Do No Harm!

Code Smells?

Code Smells identify *frequently* occurring **design problems** in a way that is more *specific or targeted* than general design guidelines (like “loosely coupled code” or “duplication-free code”). - Joshua K

A code smell is a design that duplicates, complicates, bloats or tightly couples code

A short history of Code Smells

- ☑ If it stinks, change it!
- ☑ Kent Beck coined the term code smell to signify something in code that needed to be changed.



Common Code Smells

- ☑ Inappropriate Naming
- ☑ Comments
- ☑ Dead Code
- ☑ Duplicated code
- ☑ Primitive Obsession
- ☑ Large Class
- ☑ Lazy Class
- ☑ Alternative Class with Different Interface
- ☑ Long Method
- ☑ Long Parameter List
- ☑ Switch Statements
- ☑ Speculative Generality
- ☑ Oddball Solution
- ☑ Feature Envy
- ☑ Refused Bequest
- ☑ Black Sheep
- ☑ Train Wreck



Inappropriate Naming

Inappropriate Naming

- ☑ Names given to variables (fields) and methods should be clear and meaningful.
- ☑ A variable name should say exactly what it is.
 - ☑ Which is better?
 - ☑ `private string s;` OR `private string salary;`
- ☑ A method should say exactly what it does.
 - ☑ Which is better?
 - ☑ `public double calc (double s)`
 - ☑ `public double calculateFederalTaxes (double salary)`

Comments

- ☑ Comments are often used as deodorant
- ☑ Comments represent a *failure to express an idea in the code*. Try to make your code self-documenting or intention-revealing
- ☑ When you feel like writing a comment, first try "to refactor so that the comment becomes superfluous."
- ☑ Remedies:
 - ☑ Extract Method
 - ☑ Rename Method



Comment: “Grow the Array” smells

```
public class MyList
{
    int INITIAL_CAPACITY = 10;
    bool m_readOnly;
    int m_size = 0;
    int m_capacity;
    string[] m_elements;

    public MyList()
    {
        m_elements = new string[INITIAL_CAPACITY];
        m_capacity = INITIAL_CAPACITY;
    }

    int GetCapacity() {
        return m_capacity;
    }
}
```

```
void AddToList(string element)
{
    if (!m_readOnly)
    {
        int newSize = m_size + 1;
        if (newSize > GetCapacity())
        {
            // grow the array
            m_capacity += INITIAL_CAPACITY;
            string[] elements2 = new string[m_capacity];
            for (int i = 0; i < m_size; i++)
                elements2[i] = m_elements[i];

            m_elements = elements2;
        }
        m_elements[m_size++] = element;
    }
}
```

Comment Smells Make-over

```
void AddToList(string element)
{
    if (m_readOnly)
        return;
    if (ShouldGrow())
    {
        Grow();
    }
    StoreElement(element);
}
```

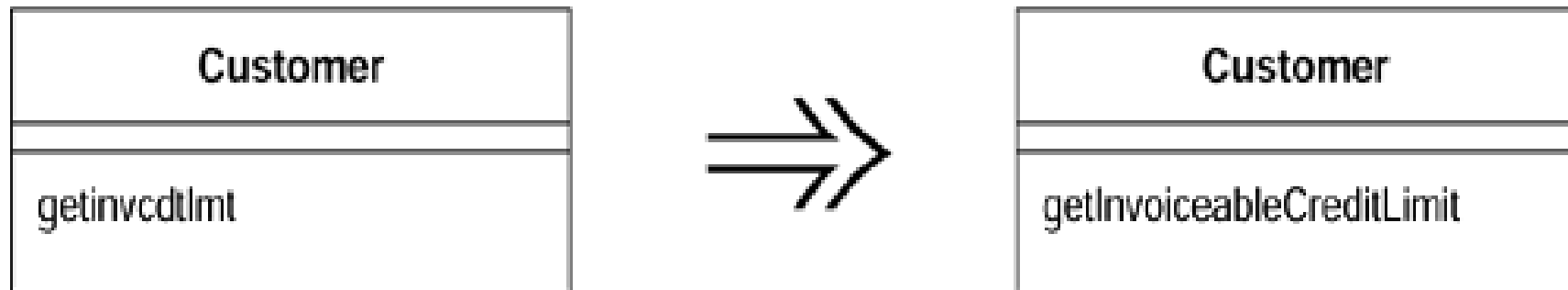
```
private bool ShouldGrow()
{
    return (m_size + 1) > GetCapacity();
}
```

```
private void Grow()
{
    m_capacity += INITIAL_CAPACITY;
    string[] elements2 = new string[m_capacity];
    for (int i = 0; i < m_size; i++)
        elements2[i] = m_elements[i];

    m_elements = elements2;
}

private void StoreElement(string element)
{
    m_elements[m_size++] = element;
}
```

Rename Method





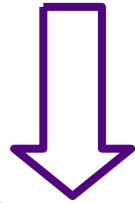
Extract Method

Extract Method

```
void PrintOwning(double amount){  
    PrintBanner();  
  
    // print details  
    System.Console.Out.WriteLine("name: "+ name);  
    System.Console.Out.WriteLine("amount: "+ amount);  
}
```

Extract Method

```
void PrintOwning(double amount){  
    PrintBanner();  
  
    // print details  
    System.Console.Out.WriteLine("name: "+ name);  
    System.Console.Out.WriteLine("amount: "+ amount);  
}
```



```
void PrintOwning(double amount){  
    PrintBanner();  
    PrintDetails(amount);  
}  
  
void PrintDetails(double amount){  
    System.Console.Out.WriteLine("name: "+ name);  
    System.Console.Out.WriteLine("amount: "+ amount);  
}
```

Long Method



Long Method

- ✓ A method is long when it is too hard to quickly comprehend.
- ✓ Long methods tend to hide behavior that ought to be shared, which leads to duplicated code in other methods or classes.
- ✓ Good OO code is easiest to understand and maintain with shorter methods with good names



Long Method

- ☑ A method is long when it is too hard to quickly comprehend.
- ☑ Long methods tend to hide behavior that ought to be shared, which leads to duplicated code in other methods or classes.
- ☑ Good OO code is easiest to understand and maintain with shorter methods with good names
- ☑ Remedies:
 - ☑ Extract Method
 - ☑ Replace Temp with Query
 - ☑ Introduce Parameter Object
 - ☑ Preserve Whole Object
 - ☑ Replace Method with Method Object
 - ☑ Decompose Conditional



Long Method Example

```
private String toStringHelper(StringBuffer result)
{
    result.append("<");
    result.append(name);
    result.append(attributes.toString());
    result.append(">");
    if (!value.equals(""))
        result.append(value);
    Iterator it = children().iterator();
    while (it.hasNext())
    {
        TagNode node = (TagNode)it.next();
        node.toStringHelper(result);
    }
    result.append("</");
    result.append(name);
    result.append(">");
    return result.toString();
}
```

Long Method Makeover (Extract Method)

```
private String toStringHelper(StringBuffer result)
{
    writeOpenTagTo(result);
    writeValueTo(result);
    writeChildrenTo(result);
    writeEndTagTo(result);
    return result.toString();
}
```

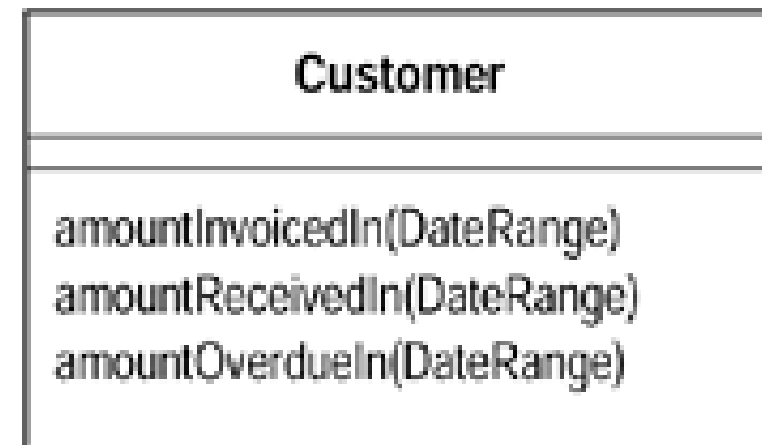
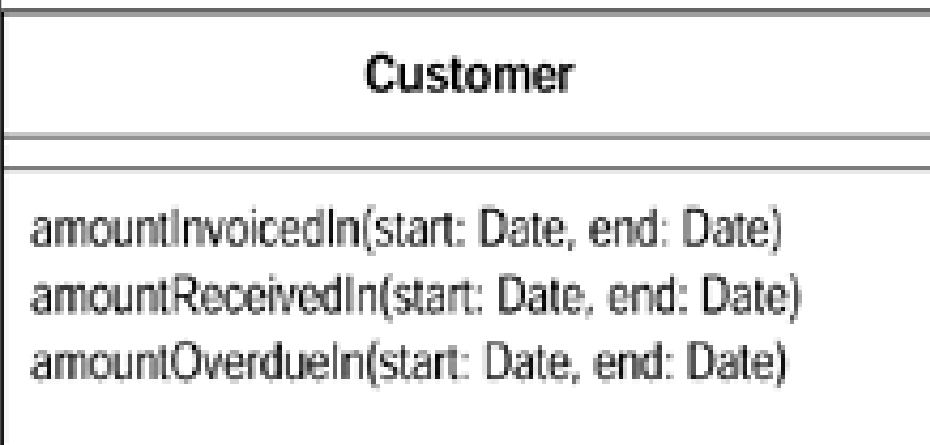
```
private void writeOpenTagTo(StringBuffer result)
{
    result.append("<");
    result.append(name);
    result.append(attributes.toString());
    result.append(">");
}
```

```
private void writeEndTagTo(StringBuffer result)
{
    result.append("</");
    result.append(name);
    result.append(">");
}
```

```
private void writeValueTo(StringBuffer result)
{
    if (!value.equals(""))
        result.append(value);
}
```

```
private void writeChildrenTo(StringBuffer result)
{
    Iterator it = children().iterator();
    while (it.hasNext())
    {
        TagNode node = (TagNode)it.next();
        node.toStringHelper(result);
    }
}
```

Introduce Parameter Object

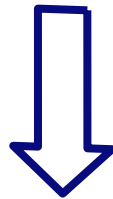


Decompose Conditional

You have a complicated conditional (if-then-else) statement.

Extract methods from the condition, then part, and else parts.

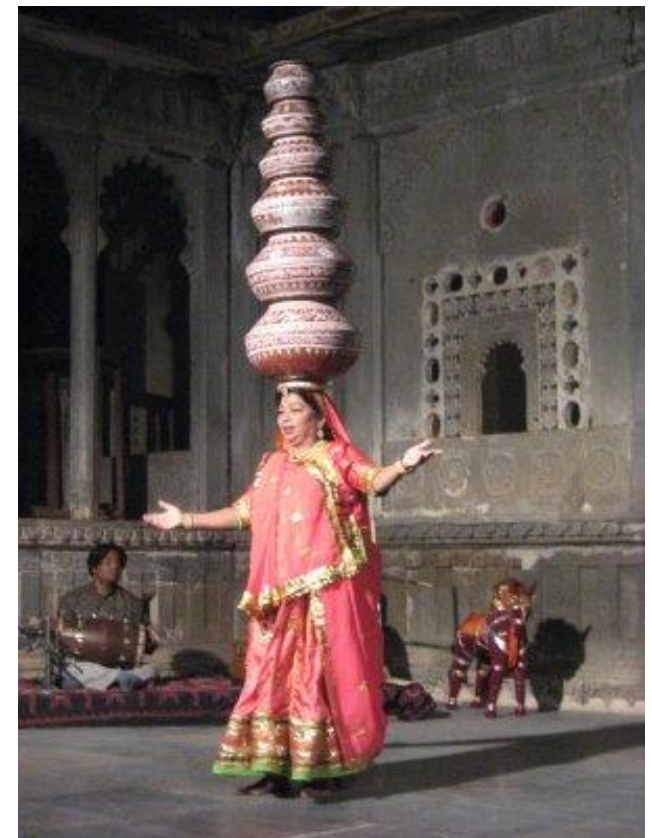
```
if (date.before (SUMMER_START) || date.after(SUMMER_END))  
    charge = quantity * _winterRate + _winterServiceCharge;  
else charge = quantity * _summerRate;
```



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

Long Parameter List

- ☑ Methods that take too many parameters produce client code that is awkward and difficult to work with.
- ☑ Remedies:
 - ☑ Introduce Parameter Object
 - ☑ Replace Parameter with Method
 - ☑ Preserve Whole Object



Example

```
private void createUserInGroup() {  
    GroupManager groupManager = new GroupManager();  
    Group group = groupManager.create(TEST_GROUP, false,  
                                     GroupProfile.UNLIMITED_LICENSES, "",  
                                     GroupProfile.ONE_YEAR, null);  
    user = userManager.create(USER_NAME, group, USER_NAME, "jack",  
                              USER_NAME, LANGUAGE, false, false, new Date(),  
                              "blah", new Date());  
}
```


Introduce Parameter Object

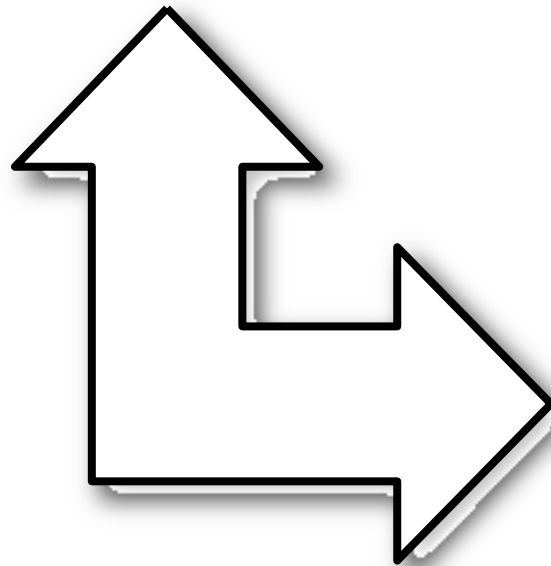
Introduce Parameter Object

Customer
AmoutInvoicedIn(Date start, Date end) AmoutRecivedIn(Date start, Date end) AmoutOverdueIn(Date start, Date end)

Introduce Parameter Object

Customer

AmoutInvoicedIn(Date start, Date end)
AmoutRecivedIn(Date start, Date end)
AmoutOverdueIn(Date start, Date end)



Customer

AmoutInvoicedIn(DateRange range)
AmoutRecivedIn(DateRange range)
AmoutOverdueIn(DateRange range)

Replace Parameter with Method

```
public double getPrice() {  
    int basePrice = _quantity * _itemPrice;  
    int discountLevel;  
    if (_quantity > 100)  
        discountLevel = 2;  
    else  
        discountLevel = 1;  
    double finalPrice = discountedPrice (basePrice, discountLevel);  
    return finalPrice;  
}
```

```
private double discountedPrice (int basePrice, int discountLevel) {  
    if (discountLevel == 2)  
        return basePrice * 0.1;  
    else  
        return basePrice * 0.05;
```

```
}
```

Replace Parameter with Method

```
public double getPrice() {  
    int basePrice = _quantity * _itemPrice;  
    int discountLevel = getDiscountLevel();  
    double finalPrice = discountedPrice (basePrice, discountLevel);  
    return finalPrice;  
}  
  
private int getDiscountLevel() {  
    if (_quantity > 100) return 2;  
    else return 1;  
}  
  
private double discountedPrice (int basePrice, int discountLevel) {  
    if (getDiscountLevel() == 2) return basePrice * 0.1;  
    else return basePrice * 0.05;  
}
```

Replace Parameter with Method

```
public double getPrice() {  
    int basePrice = _quantity * _itemPrice;  
    int discountLevel = getDiscountLevel();  
    double finalPrice = discountedPrice (basePrice);  
    return finalPrice;  
}  
  
private double discountedPrice (int basePrice) {  
    if (getDiscountLevel() == 2) return basePrice * 0.1;  
    else return basePrice * 0.05;  
}
```

Feature Envy

- ☑ A method that seems more interested in some other class than the one it is in.
- ☑ Data and behavior that acts on that data belong together. When a method makes too many calls to other classes to obtain data or functionality, Feature Envy is in the air.
- ☑ Remedies:
 - ☑ Move Field
 - ☑ Move Method
 - ☑ Extract Method



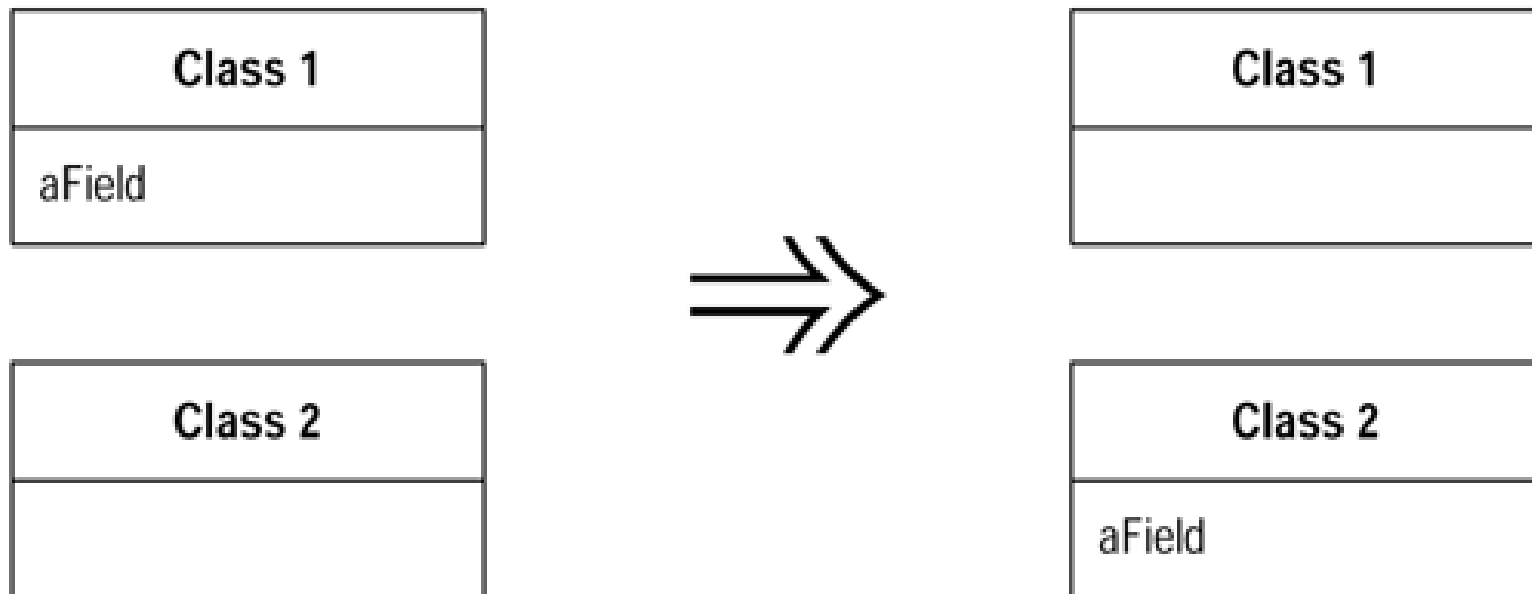
Example

```
Public class CapitalStrategy{
    double capital(Loan loan)
    {
        if (loan.getExpiry() == NO_DATE && loan.getMaturity() != NO_DATE)
            return loan.getCommitmentAmount() * loan.duration() * loan.riskFactor();

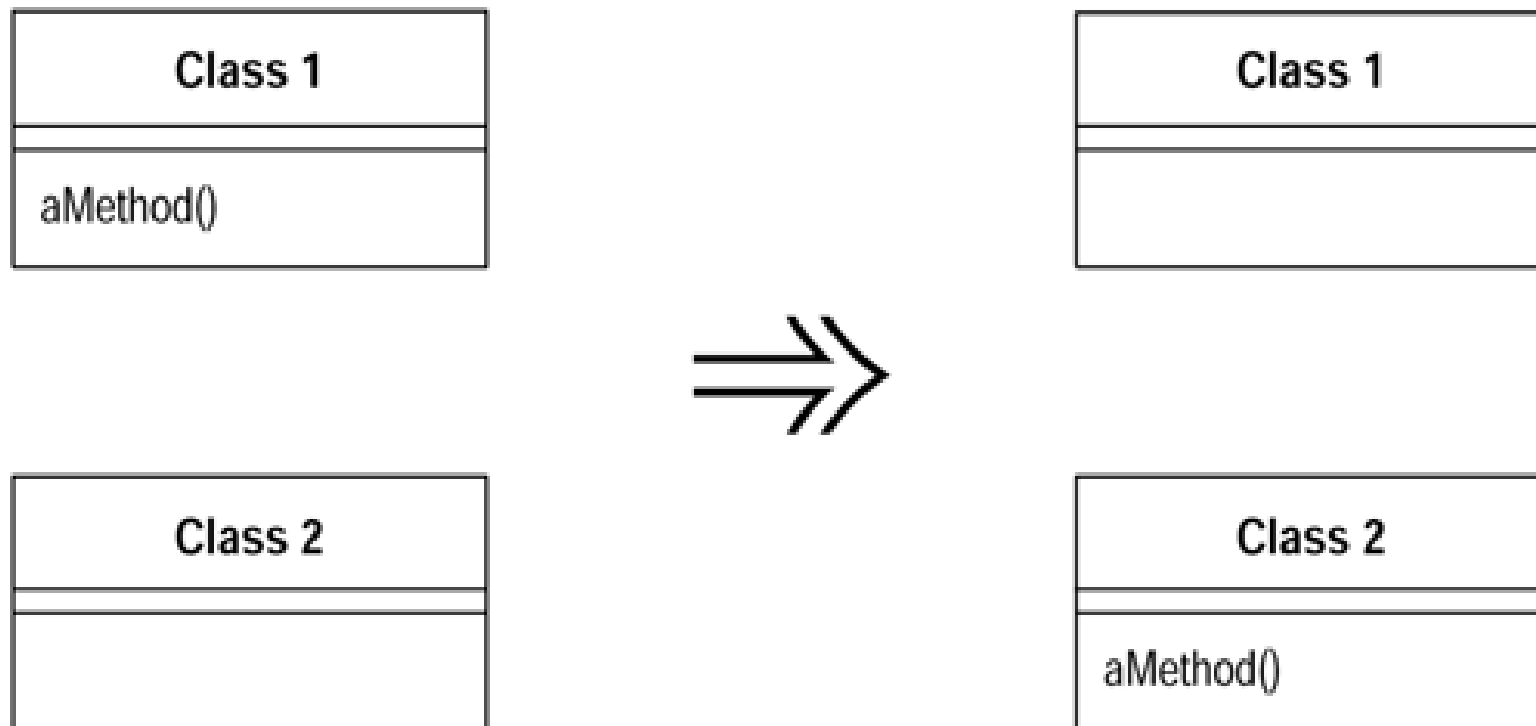
        if (loan.getExpiry() != NO_DATE && loan.getMaturity() == NO_DATE)
        {
            if (loan.getUnusedPercentage() != 1.0)
                return loan.getCommitmentAmount() * loan.getUnusedPercentage() *
loan.duration() * loan.riskFactor();
            else
                return (loan.outstandingRiskAmount() * loan.duration() * loan.riskFactor()) +
                    (loan.unusedRiskAmount() * loan.duration() * loan.unusedRiskFactor());
        }

        return 0.0;
    }
}
```


Move Field



Move Method



Dead Code

☑ Code that is no longer used in a system or related system is Dead Code.

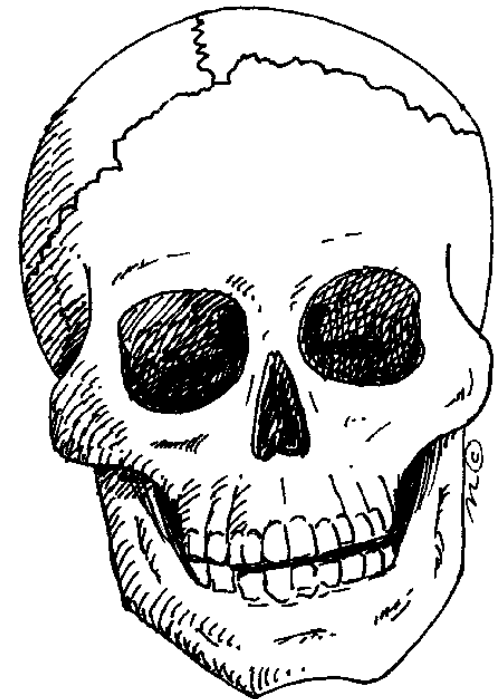
☑ Increased Complexity.

☑ Accidental Changes.

☑ More Dead Code

☑ Remedies

☑



Dead Code Example

A Loan class contains five constructors, three of which are shown below:

```
public class Loan...
public Loan(double commitment, int riskRating, Date maturity, Date expiry) { this(commitment, 0.00,
riskRating, maturity, expiry); }

public Loan(double commitment, double outstanding, int customerRating, Date maturity, Date expiry) {
this(null, commitment, outstanding, customerRating, maturity, expiry); }

public Loan(CapitalStrategy capitalStrategy, double commitment, int riskRating, Date maturity, Date expiry) {
this(capitalStrategy, commitment, 0.00, riskRating, maturity, expiry); } ... }
```

One of the above constructors is never called by a client. It is dead code.

Duplicated Code

- ☑ The *most pervasive and pungent smell* in software
- ☑ There is obvious or blatant duplication
 - ☑ Such as copy and paste
- ☑ There are subtle or non-obvious duplications
 - ☑ Such as parallel inheritance hierarchies.
 - ☑ Similar algorithms
- ☑ Remedies
 - ☑ Extract Method
 - ☑ Pull Up Field
 - ☑ Form Template Method
 - ☑ Substitute Algorithm

Ctl+C Ctl+V Pattern

```
public static MailTemplate getStaticTemplate(Languages language) {
    MailTemplate mailTemplate = null;
    if(language.equals(Languages.English)) {
        mailTemplate = new EnglishLanguageTemplate();
    } else if(language.equals(Languages.French)) {
        mailTemplate = new FrenchLanguageTemplate();
    } else if(language.equals(Languages.Chinese)) {
        mailTemplate = new ChineseLanguageTemplate();
    } else {
        throw new IllegalArgumentException("Invalid language type specified");
    }
    return mailTemplate;
}

public static MailTemplate getDynamicTemplate(Languages language, String content) {
    MailTemplate mailTemplate = null;
    if(language.equals(Languages.English)) {
        mailTemplate = new EnglishLanguageTemplate(content);
    } else if(language.equals(Languages.French)) {
        mailTemplate = new FrenchLanguageTemplate(content);
    } else if(language.equals(Languages.Chinese)) {
        mailTemplate = new ChineseLanguageTemplate(content);
    } else {
        throw new IllegalArgumentException("Invalid language type specified");
    }
    return mailTemplate;
}
```

```

private void AddOrderMaterials(int iOrderId)
{

    if (iOrderType == 1)
    {
        OrderMaterial oOrderMaterialCoffee = new OrderMaterial();
        oOrderMaterialCoffee.MaterialId = 1;
        oOrderMaterialCoffee.OrderId = iOrderId;
        oOrderMaterialCoffee.Quantity = 2;
        oDataContext.OrderMaterials.Inserton<x>submit(oOrderMaterialCoffee);

        oDataContext.SubmitChanges();
    }
    else if (iOrderType == 2)
    {
        OrderMaterial oOrderMaterialCoffee = new OrderMaterial();
        oOrderMaterialCoffee.MaterialId = 1;
        oOrderMaterialCoffee.OrderId = iOrderId;
        oOrderMaterialCoffee.Quantity = 2;
        oDataContext.OrderMaterials.Inserton<x>submit(oOrderMaterialCoffee);

        OrderMaterial oOrderMaterialCream = new OrderMaterial();
        oOrderMaterialCream.MaterialId = 2;
        oOrderMaterialCream.OrderId = iOrderId;
        oOrderMaterialCream.Quantity = 2;
        oDataContext.OrderMaterials.Inserton<x>submit(oOrderMaterialCream);

        oDataContext.SubmitChanges();
    }
    else if (iOrderType == 3)
    {
        OrderMaterial oOrderMaterialCoffee = new OrderMaterial();
        oOrderMaterialCoffee.MaterialId = 1;
        oOrderMaterialCoffee.OrderId = iOrderId;
        oOrderMaterialCoffee.Quantity = 2;
        oDataContext.OrderMaterials.Inserton<x>submit(oOrderMaterialCoffee);

        OrderMaterial oOrderMaterialSugar = new OrderMaterial();
        oOrderMaterialSugar.MaterialId = 3;
        oOrderMaterialSugar.OrderId = iOrderId;
        oOrderMaterialSugar.Quantity = 2;
        oDataContext.OrderMaterials.Inserton<x>submit(oOrderMaterialSugar);

        oDataContext.SubmitChanges();
    }
    else if (iOrderType == 4)

```



Levels of Duplication



Literal Duplication

Same for loop in 2 places

Semantic Duplication

1stLevel - For and For Each Loop

```
stack.push(1);  
stack.push(3);  
stack.push(5);  
stack.push(10);  
stack.push(15);
```

v/s

2ndLevel - Loop v/s Lines repeated

```
for(int i : asList(1,3,5,10,15))  
stack.push(i);
```



Data Duplication

Some constant declared in 2 classes (test and production)



Conceptual Duplication

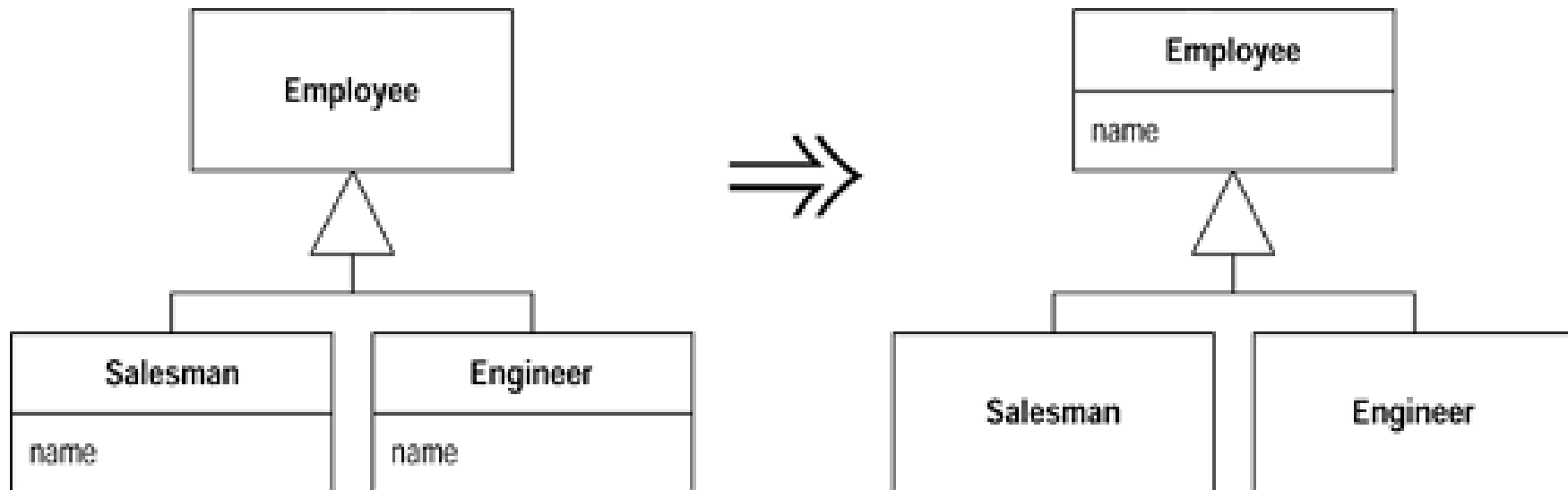
2 Algorithm to Sort elements (Bubble sort and Quick sort)

Logical Steps - Duplication

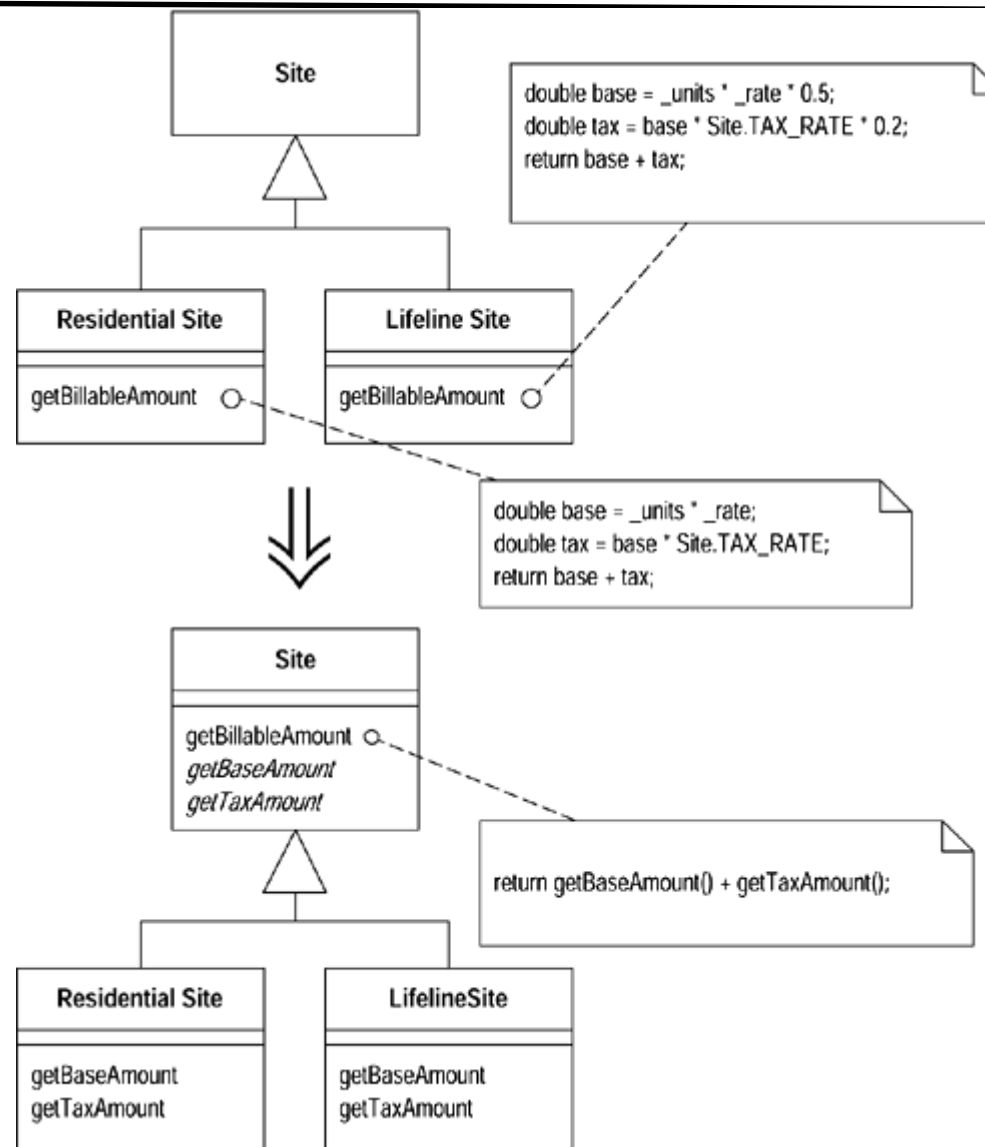
Same set of steps repeat in different scenarios.

Ex: Same set of validations in various points in your applications

Pull Up Field



Form Template Method



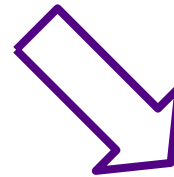
Substitute Algorithm

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 ""; }  
}
```

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 "";  
}
```



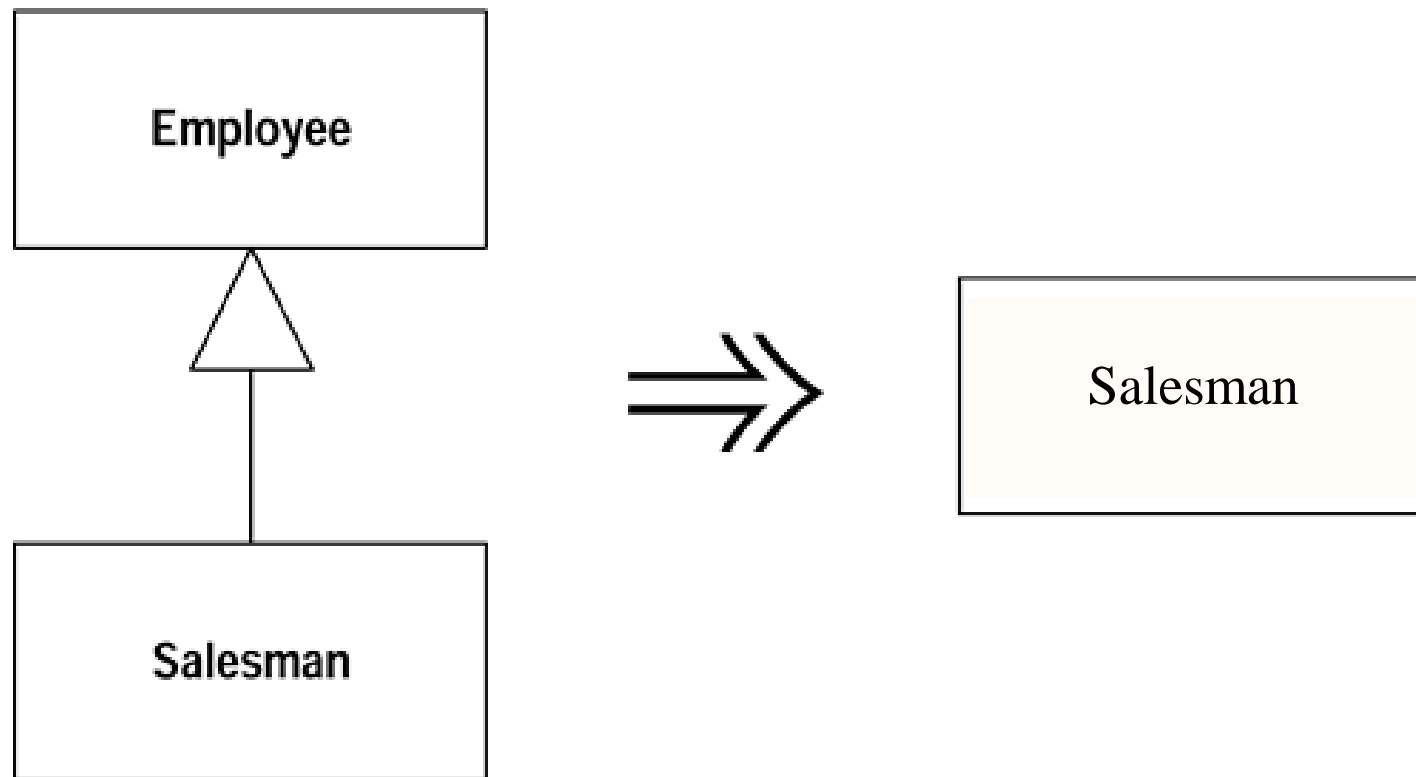
```
String foundPerson(String[] people){  
    List candidates = Arrays.asList(new String[] {"Don",  
    "John", "Kent"});  
    for (String person : people)  
        if (candidates.contains(person))  
            return person;  
    return "";  
}
```

Speculative Generality

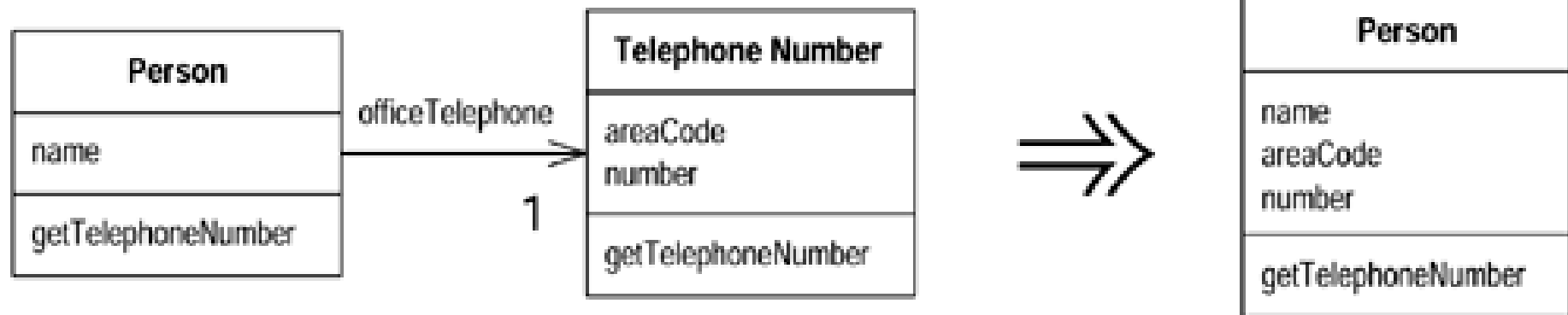
- ☑ You get this smell when people say "Oh, I think we will need the ability to do that someday" and thus want all sorts of hooks and special cases to handle things that aren't required.
- ☑ This odor exists when you have generic or abstract code that isn't actually needed today. Such code often exists to support future behavior, which may or may not be necessary in the future.
- ☑ Remedies
 - ☑ Collapse Hierarchy
 - ☑ Inline Class
 - ☑ Remove Parameter



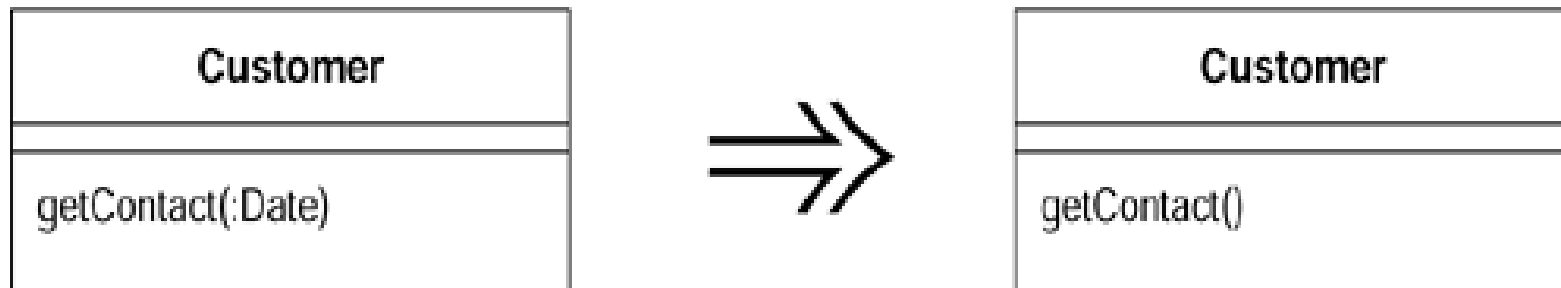
Collapse Hierarchy



Inline Class



Remove Parameter

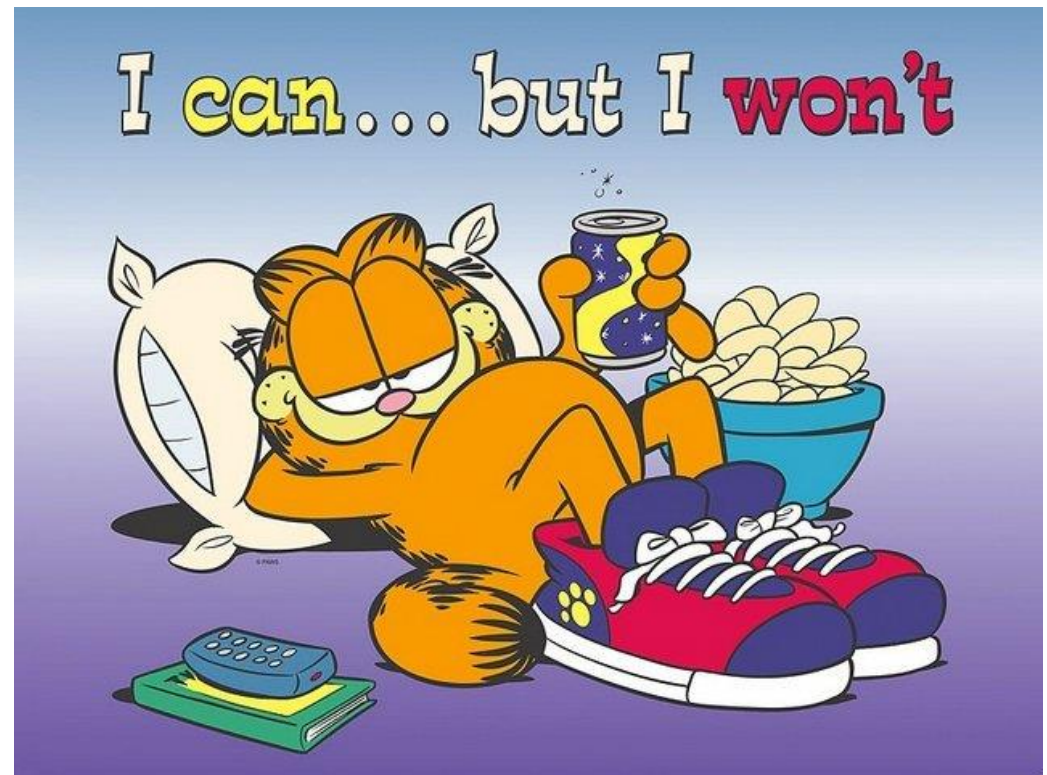


Lazy Class

- ☑ A class that isn't doing enough to carry its weight
- ☑ We let the class die with dignity
- ☑ Often this might be a class that used to pay its way but has been downsized with refactoring. Or it might be a class that was added because of changes that were planned but not made.

Remedies

- ☑ Inline Class
- ☑ Collapse Hierarchy



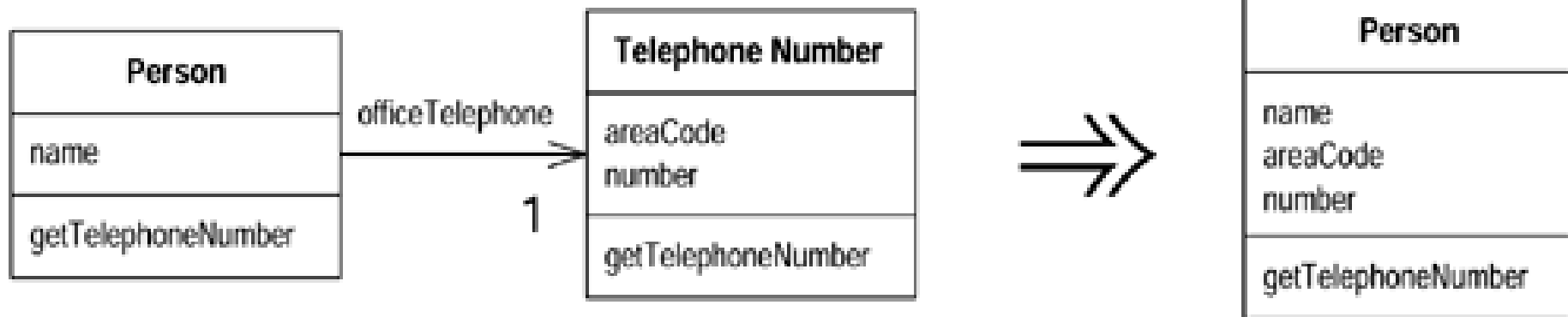
Lazy Class Example

```
public interface SomeInterface {  
    void methodOne();  
    void defaultMethod();  
}  
public abstract class LazyClazz implements SomeInterface {  
    public abstract void methodOne();  
    public void defaultMethod() {  
        //do nothing  
    }  
}  
public class WorkerClazz extends LazyClazz {  
    public void methodOne() {  
        // some actual code here  
    }  
    public void defaultMethod() {  
        //some more actual code  
    }  
}
```

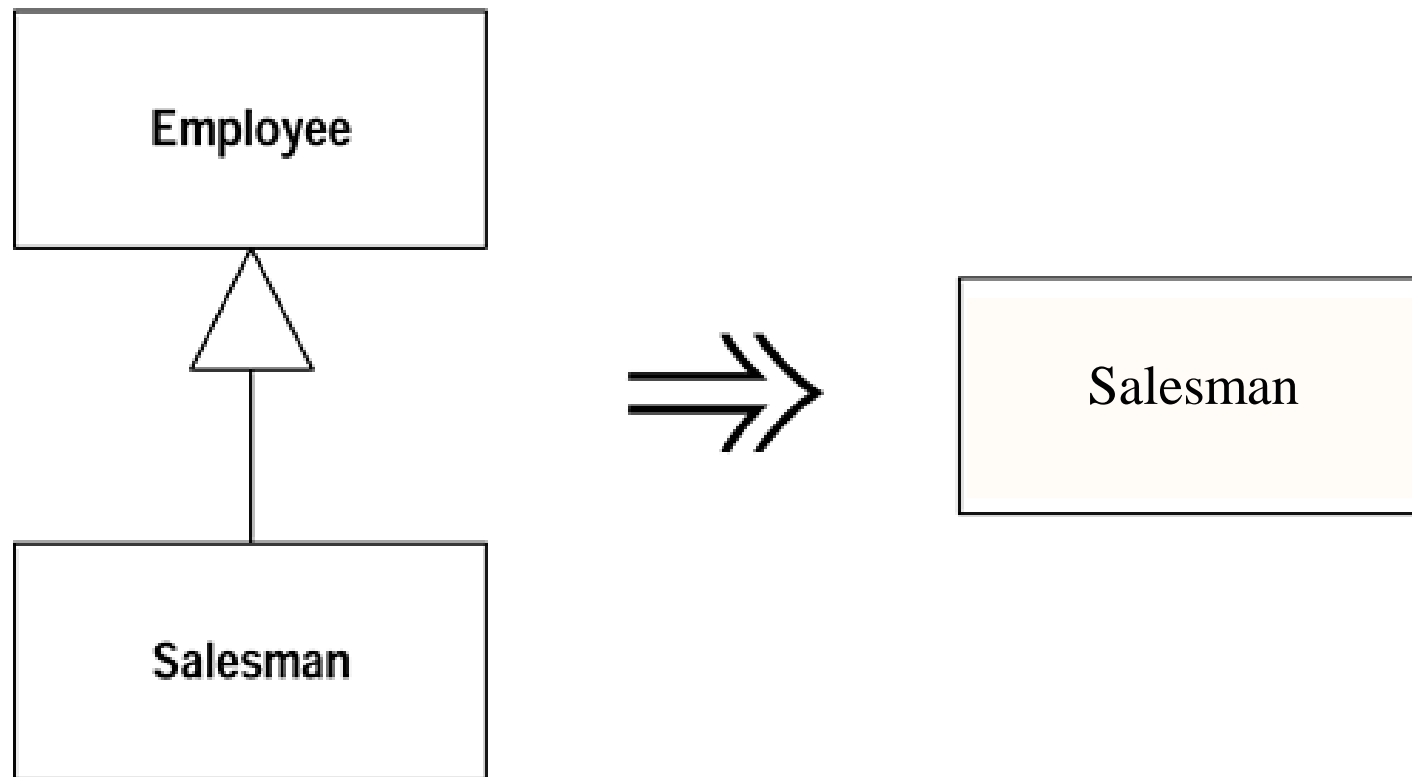

Another Lazy Class

```
public class Letter {  
    private final String content;  
  
    public Letter(String content) {  
        this.content = content;  
    }  
  
    public String getContent() {  
        return content;  
    }  
}
```

Inline Class



Collapse Hierarchy



Black Sheep

- ☑ Sometimes a subclass or method *doesn't fit* in so well with its *family*.
- ☑ A subclass that is substantially different in nature than other subclasses in the hierarchy.
- ☑ A method in a class that is noticeably different from other methods in the class.



Example

```
public class StringUtil {  
    public static String pascalCase(String string) {  
        return string.substring(0,1).toUpperCase() + string.substring(1);  
    }  
  
    public static String camelCase(String string) {  
        return string.substring(0,1).toLowerCase() + string.substring(1);  
    }  
  
    public static String numberAndNoun(int number, String noun) {  
        return number + " " + noun + (number != 1 ? "s" : "");  
    }  
  
    public static String extractCommandNameFrom(Map parameterMap) {  
        return ((String[]) parameterMap.get("command"))[0];  
    }  
}
```

Primitive Obsession

- ☑ This smell exists when primitives, such as strings, doubles, arrays or low-level language components, are used for high-level operations instead of using classes.
- ☑ This typically occurs when you haven't yet seen how a higher-level abstraction can clarify or simplify your code.
- ☑ Remedies
 - ☑ Extract Class
 - ☑ Replace Data Value with Object
 - ☑ Replace Type Code with Class
 - ☑ Introduce Parameter Object
 - ☑ Replace Array with Object



Primitive Obsession Example

```
if (someString.indexOf("substring") != -1)
```



```
if(someString.contains("substring"))
```

Primitive Obsession Example

```
private void Grow() {  
    Object[] newElements = new Object[elements.length + 10];  
    for (int i = 0; i < size; i++)  
        newElements[i] = elements[i];  
  
    elements = newElements;  
}
```



```
private void Grow() {  
    Object[] newElements = new Object[elements.length + INITIAL_CAPACITY];  
    System.arraycopy(elements, 0, newElements, 0, size);  
    elements = newElements;  
}
```


Primitive Obsession Example

```
public class CompositeShape
{
    IShape [] arr = new IShape[100];
    int count = 0;

    public void Add(IShape shape){
        arr[count++] = shape;
    }

    public void Remove(IShape shape)
    {
        for (int i = 0; i < 100; i++)
        {
            if (shape == arr[i])
            {
                //code to remove
            }
        }
    }
}
```

Primitive Obsessed Code - Make Over

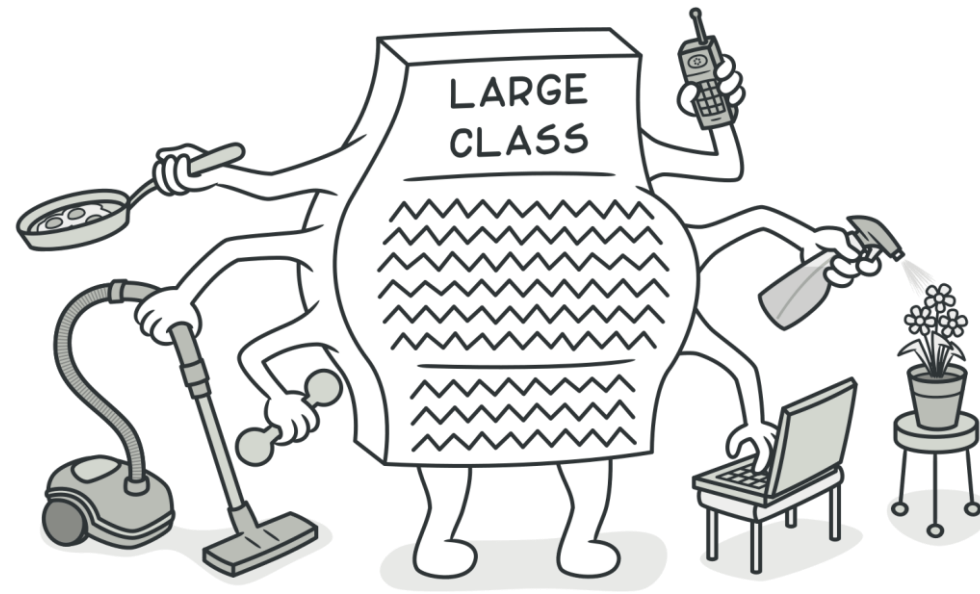
```
public class CompositeShape
{
    List<IShape> shapeList = new List<IShape>();

    public void Add(IShape shape)
    {
        shapeList.Add(shape);
    }

    public void Remove(IShape shape)
    {
        shapeList.Remove(shape);
    }
}
```

Large Class

- ☑ Like people, classes suffer when they take on too many responsibilities.
- ☑ GOD Objects
- ☑ Fowler and Beck note that the presence of too many instance variables usually indicates that a class is trying to do too much. In general, large classes typically contain too many responsibilities.
- ☑ Remedies
 - ☑ Extract Class
 - ☑ Replace Type Code with Class/Subclass
 - ☑ Replace Type Code with State/Strategy
 - ☑ Replace Conditional with Polymorphism



```

public Vector clearWorkspaceAfterPayrollGeneration(long payrollId, String dbName,
String username, String password, String debug) {
    long returnCode = 201;
    String errorCode = "Uncaught exception";
    PayrollDB db = null;
    Connection conn = null;
    Payroll payroll = null;
    PayrollProject payrollProject = null;
    WorkspaceUtil wsu = new WorkspaceUtil();
    Workspace ws = null;
    try {
        for (int i = 0; i < 11; i++) {
            ws = wsu.getWorkspace();
            if (ws == null) {
                returnCode = 101;
                errorCode = "unable to connect to workspace";
                ws.logMessage(errorCode + "\n", true);
                break;
            }
            java.util.Date curTime = new java.util.Date(); // get current
            // time
            ws.logMessage("\n-----" + curTime
                + "-----\n", true);
            ws.logMessage("clearWorkspaceAfterPayrollGeneration(" + payrollId + "," + dbName
                + "," + username + ",*****)\n", true);
            errorCode = "Creating PayrollDb object";
            db = new PayrollDB(username, password, dbName);
            errorCode = "Connecting to database";
            conn = db.getConnection();
            errorCode = "Converting payrollId to Integer";
            Integer iPayrollId = new Integer((int) payrollId);
            errorCode = "Creating Payroll object";
            payroll = new Payroll(db);
            errorCode = "Calling payroll.selectRowById";
            if (!(payroll.selectRowById(iPayrollId))) {
                returnCode = 102;
                errorCode = "selectPayroll(" + iPayrollId + ") failed";
                ws.logMessage(errorCode + "\n", true);
                break;
            }
            errorCode = "PayrollName (" + payroll.getPayrollName() + ") PayrollType ("
                + payroll.getPayrollType() + ")";
            if (debug.equals("Y"))
                ws.logMessage(errorCode + "\n", true);
            errorCode = "Creating payrollProject object";
            payrollProject = new PayrollProject(db);
            errorCode = "Calling payrollProject.selectRowsById";
            if (!(payrollProject.selectRowsById(iPayrollId))) {
                returnCode = 103;
                errorCode = "selPayrollProject(" + iPayrollId + ") failed";
                ws.logMessage(errorCode + "\n", true);
                break;
            }
            errorCode = "Unloading (" + payrollProject.getRowCnt() + " projects found)";
            boolean bRowFound = payrollProject.firstRow();
            String projectName = null;
            String projectVersion = null;
            Boolean payrollProjectFlag = null;
            long lUnloadCnt = 0;
            OrganizationProjects paygej = null;

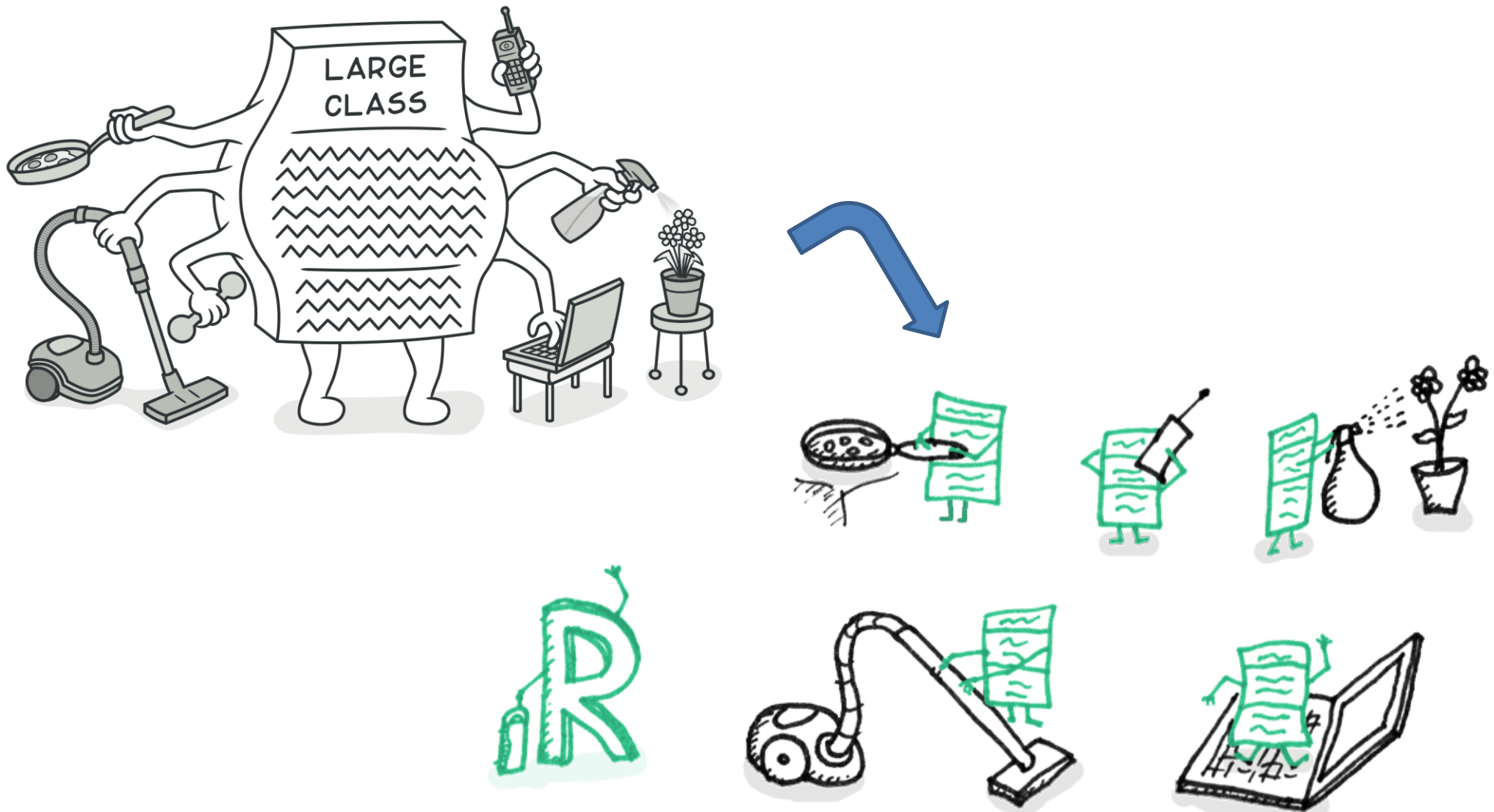
```

```

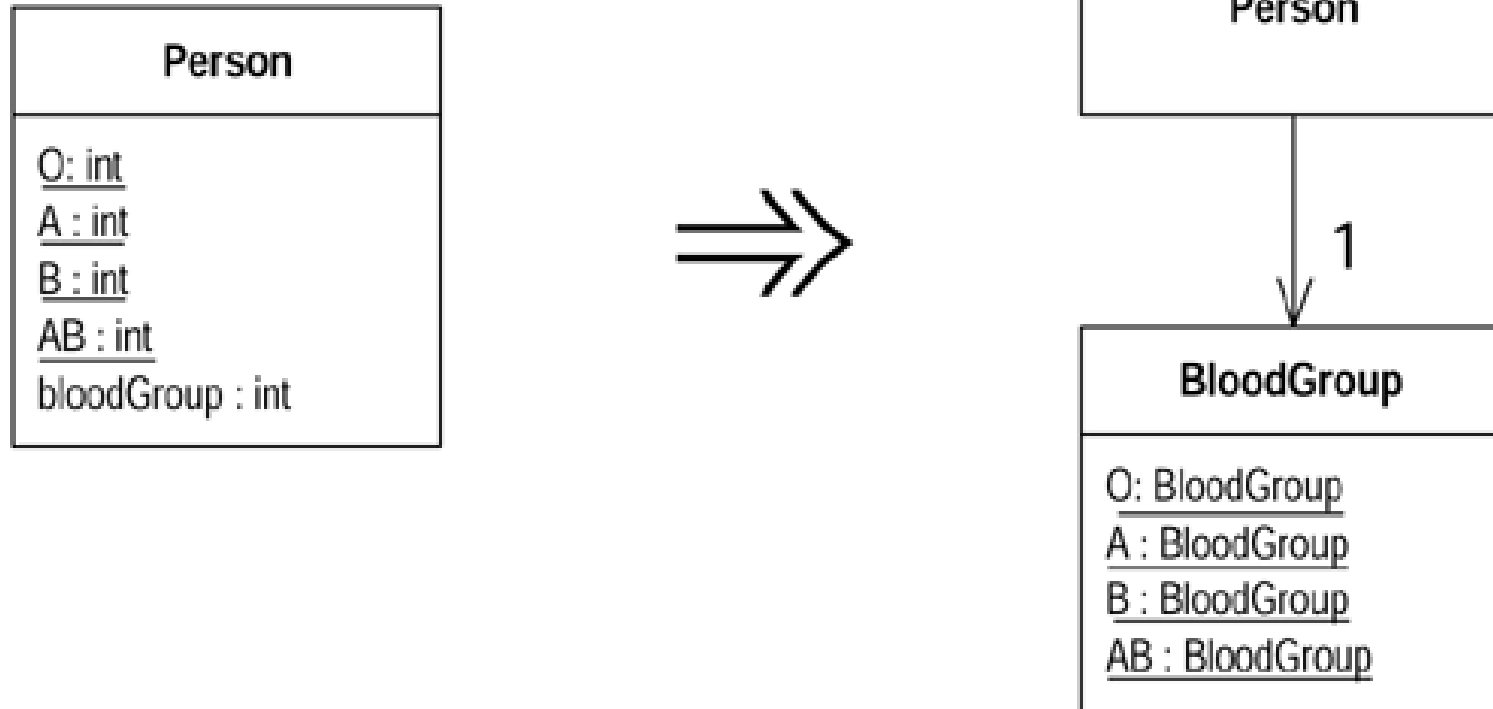
while (bRowFound) {
    projectName = payrollProject.getProjectName();
    projectVersion = payrollProject.getProjectVersion();
    payrollProjectFlag = payrollProject.getPayrollProjectFlag();
    errorCode = "-UnloadingProject (" + projectName + ") Ver (" + projectVersion
        + ") PayrollProject (" + payrollProjectFlag + ")";
    try {
        paygej = new OrganizationProjects(projectName);
    } catch (Exception e) {
        paygej = null;
        errorCode = errorCode + " (non-standard project ignored)";
    }
    if (paygej != null) {
        if (paygej.isOrganizationWideProject() && paygej.isNonPayrollProject()) {
            errorCode = errorCode + " (bypassed common project)";
        } else {
            if (paygej.isATPPProject()) {
                errorCode = errorCode + " unloadATPPProject";
                lUnloadCnt = wsu.unloadATPPProject(paygej, null);
            } else {
                errorCode = errorCode + " unloadProject";
                if (wsu.unloadProject(projectName))
                    lUnloadCnt = 1;
                else
                    lUnloadCnt = 0;
            }
            errorCode = errorCode + " (unloaded " + lUnloadCnt + " projects)";
        }
    }
    if (debug.equals("Y"))
        ws.logMessage(errorCode + "\n", true);
    bRowFound = payrollProject.nextRow();
}
returnCode = 0;
errorCode = "Success";
ws.logMessage(errorCode + "\n", true);
}
} catch (Exception e) {
    errorCode = errorCode + ": excp(" + e + ")";
    if (ws != null) {
        try {
            ws.logMessage(errorCode + "\n", true);
        } catch (Exception eee) {
            errorCode = errorCode + " LOG FAILED: excp(" + eee + ")";
        }
    }
} finally {
    if (db != null) {
        try {
            db.closeConnection();
        } catch (Exception ee) {
            // ignore
        }
    }
}
return commandUtil.commandVector(returnCode, errorCode);
}

```

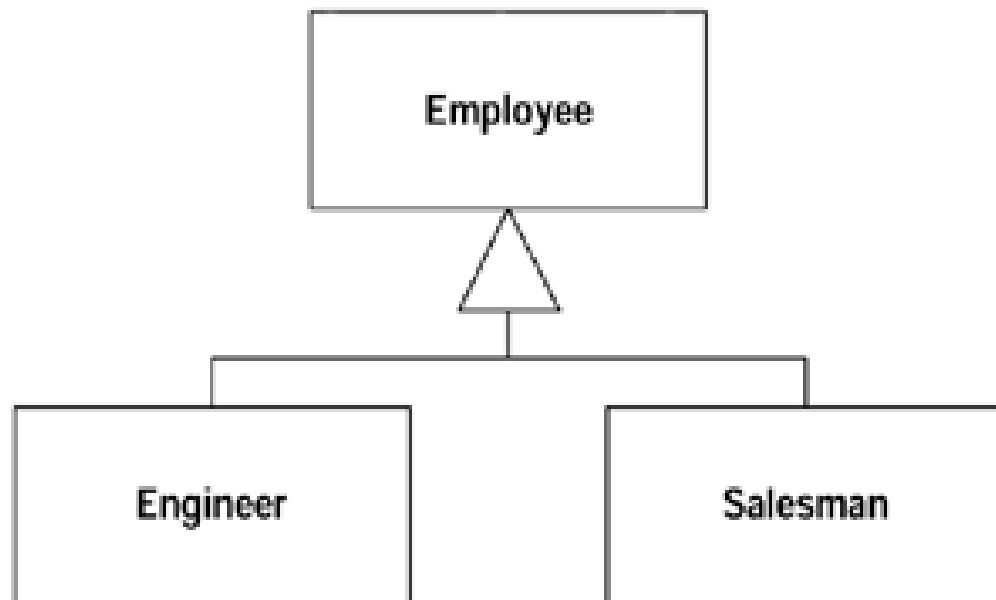
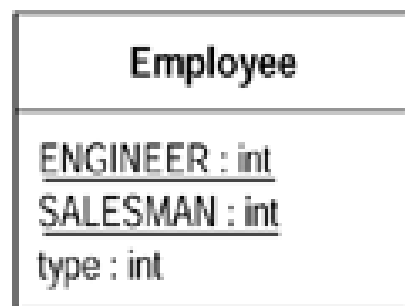
Extract Class



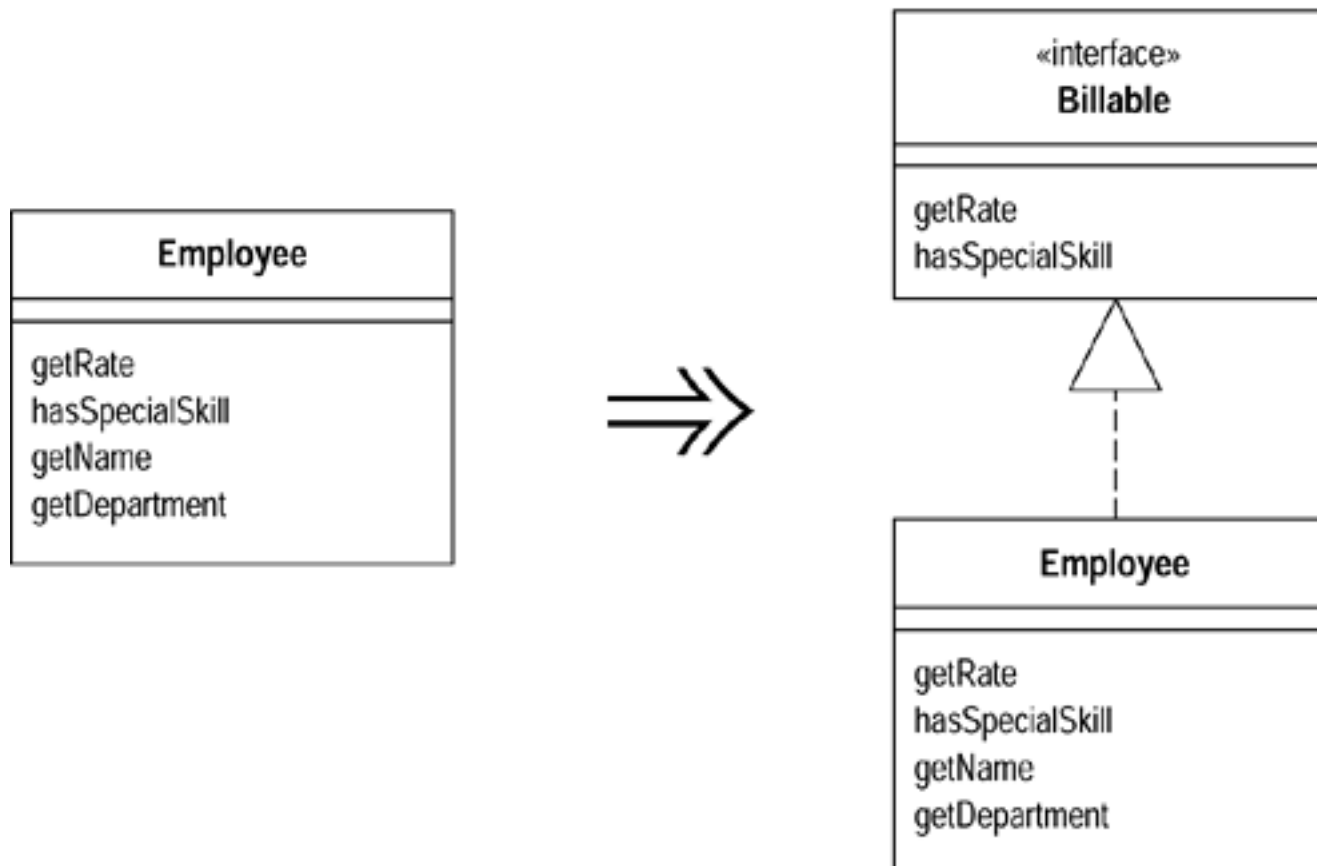
Replace Type Code with Class



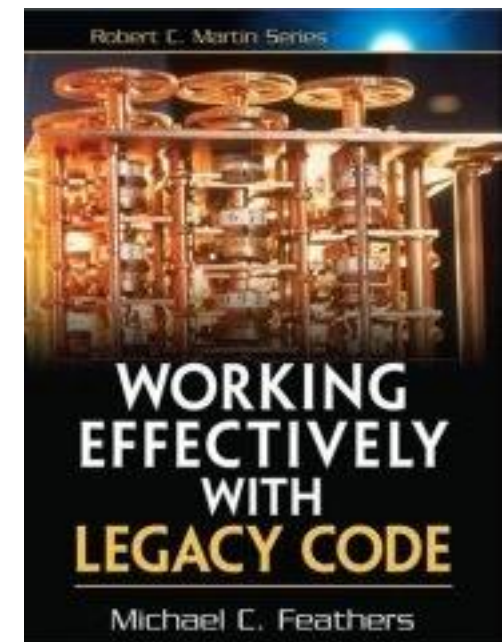
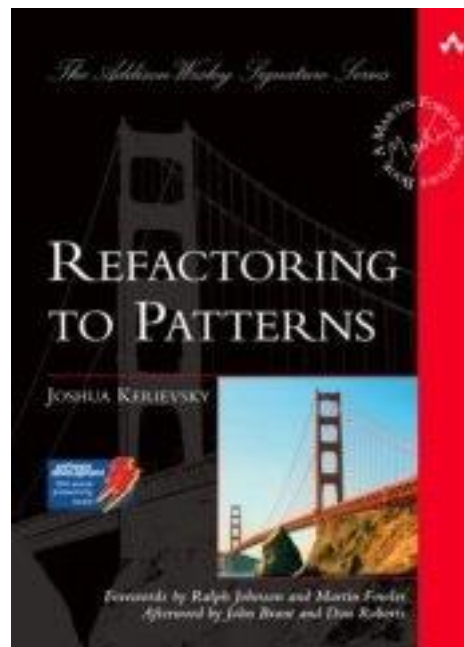
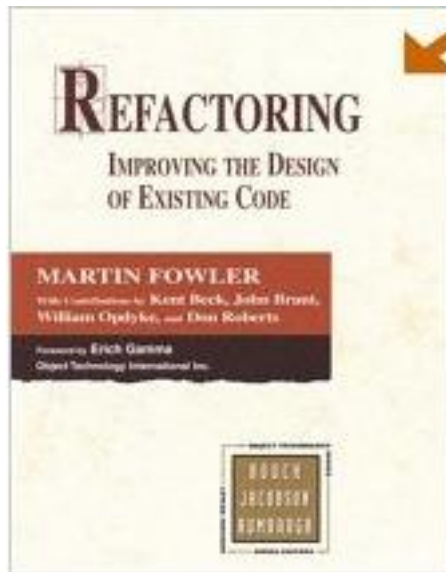
Replace Type Code with Subclasses



Extract (Narrow) Interface



Reference Reading



Further Information On Code Smells and Refactoring

- ☑ Wiki Discussion About Code Smells: <http://c2.com/cgi/wiki?CodeSmell>
- ☑ Mika's Smell Taxonomy: <http://www.soberit.hut.fi/mmantyla/BadCodeSmellsTaxonomy.htm>
- ☑ Bill Wake's book, "Refactoring Workbook"
- ☑ Refactoring Catalog Online: <http://www.refactoring.com/catalog/index.html>
- ☑ Refactoring to Patterns Catalog Online: <http://industriallogic.com/xp/refactoring/catalog.html>