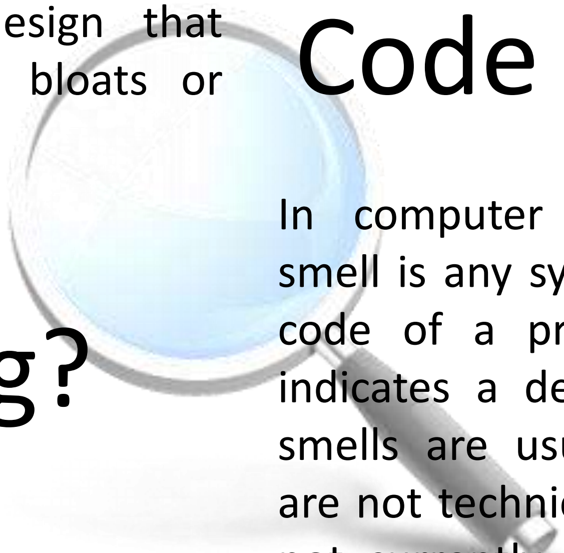
Day 3

Code Smells and Refactoring

Photo Source : Industrial Logic



Code Smells?

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

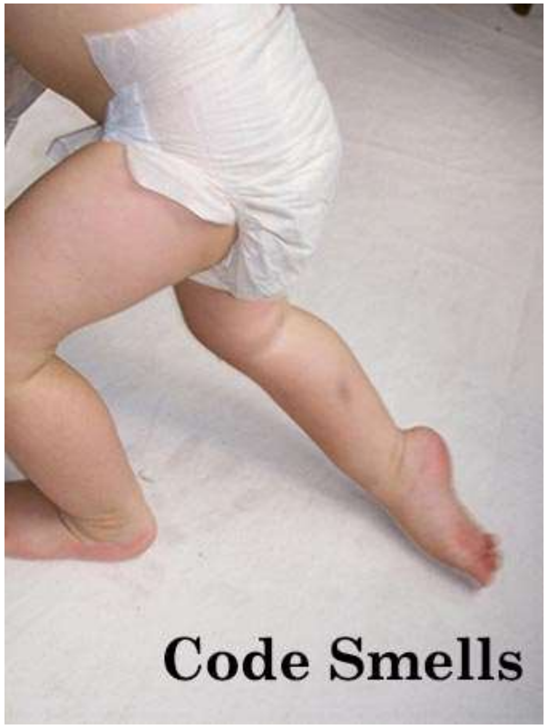
In computer programming, code smell is any symptom in the source code of a program that possibly indicates a deeper problem. Code smells are usually not bugs—they are not technically incorrect and do not currently prevent the program from functioning. Instead, they indicate weaknesses in design that may be slowing down development or increasing the risk of bugs or failures in the future.

Refactoring?

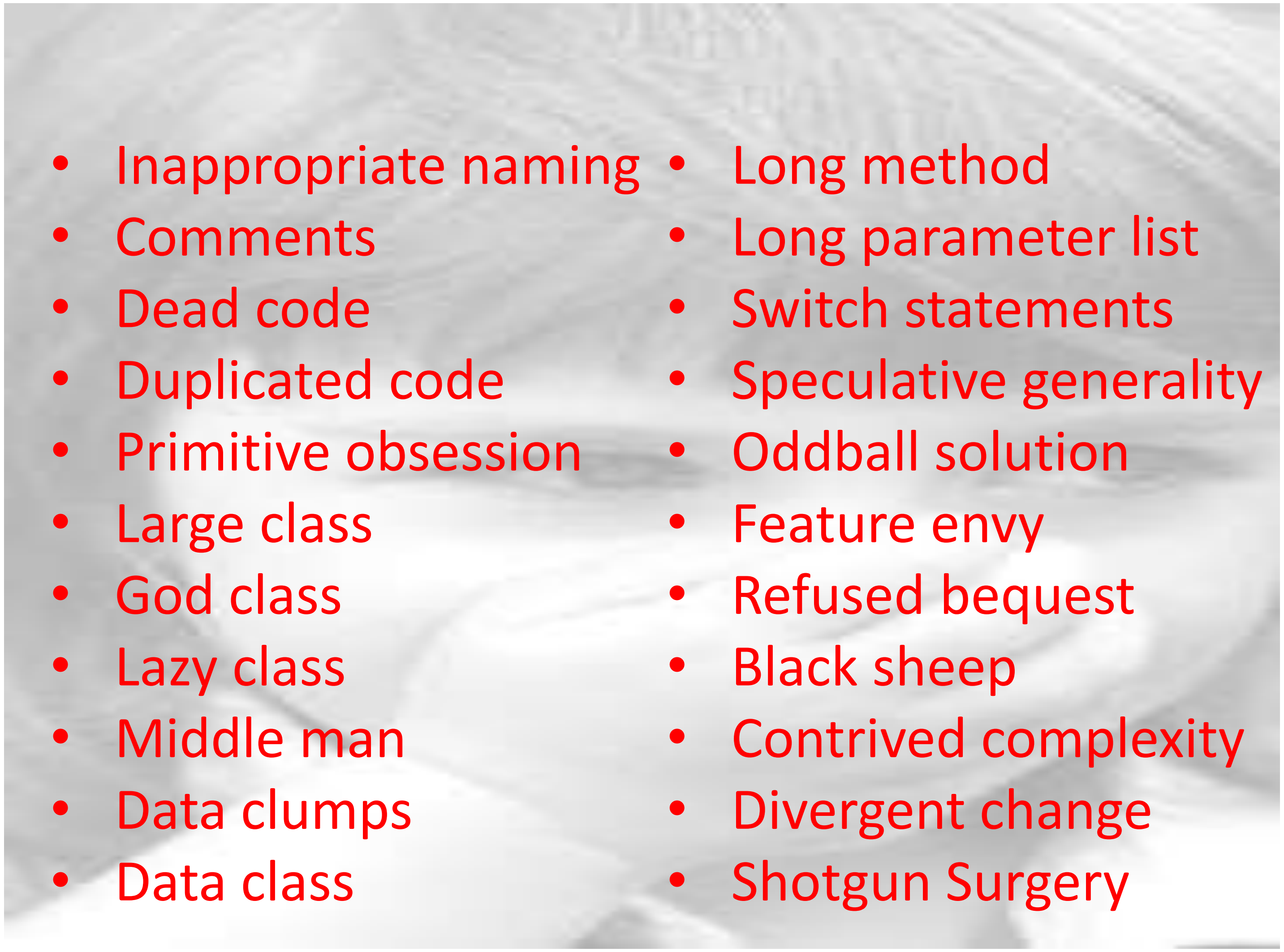
"Refactoring is the process of changing a software system in such a way that it does not alter the external behavior of the code yet improves its internal structure." – Martin Fowler

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
- God class
- Lazy class
- Middle man
- Data clumps
- Data class
- Long method
- Long parameter list
- Switch statements
- Speculative generality
- Oddball solution
- Feature envy
- Refused bequest
- Black sheep
- Contrived complexity
- Divergent change
- Shotgun Surgery

Inappropriate Naming

- Names given to variables (fields) ,methods or class should be clear and meaningful.
- A variable, field, class 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); OR

  public double calculateFederalTaxes (double salary);

Remedy:

- ❑Rename Variables, Fields, Method, Class

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 it.

- Remedy:
  - ❑ Extract Method
  - ❑ Rename Method

Comments (Cont'd)

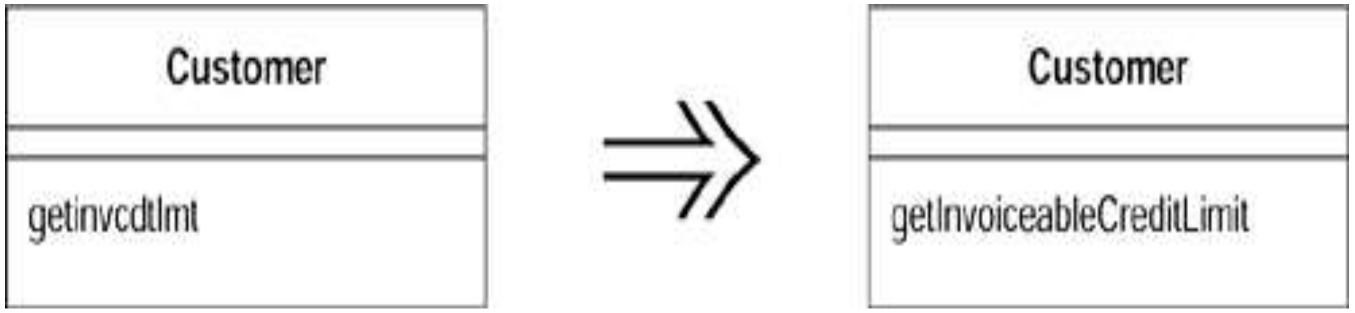
```
void List::add(string element)
{
 if (!m_readOnly)
{
       int newSize = m_size + 1;
        if (newSize > getCapacity())
       {
             // grow the array
             m_capacity += INITIAL_CAPACITY;
             string* elements2 = new string[m_capacity];
             for (int i = 0; i < m_size; i++)
             elements2[i] = m_elements[i];
             delete[] m_elements;
             m_elements = elements2;
      }
       m_elements[m_size++] = element;
}
```

Comments (Cont'd)

```
void List::add(string element)
{
if (m_readOnly)
   return;
if (shouldGrow())
   grow();
   storeElement(element);
 }

bool List::shouldGrow()
{
   return (m_size + 1) >
capacity();
}
```

```
void List::grow()
{
   m_capacity += 10;
   string *newElements = new string[m_capacity];
   for(int i = 0;i < m_size;i++)
      newElements[i] = m_elements[i];
   delete [] m_elements;
   m_elements = newElements;
}

void List::storeElement(string element)
{
    m_elements[m_size++] = element;
}
```

Rename Method



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

- 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 (Cont'd)

```
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();
}
```

Remedy of Long method

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 writeValueTo(StringBuffer result) {
      if (!value.equals(""))
            result.append(value);
}
private void writeChildrenTo(StringBuffer result) {
       Iterator it = children().iterator();
       while (it.hasNext()) {
              T agNode node = (TagNode)it.next();
               node.toStringHelper(result);
       }
}
private void writeEndTagTo(StringBuffer result)
{
        result.append("</");
        result.append(name);
        result.append(">");
}
```

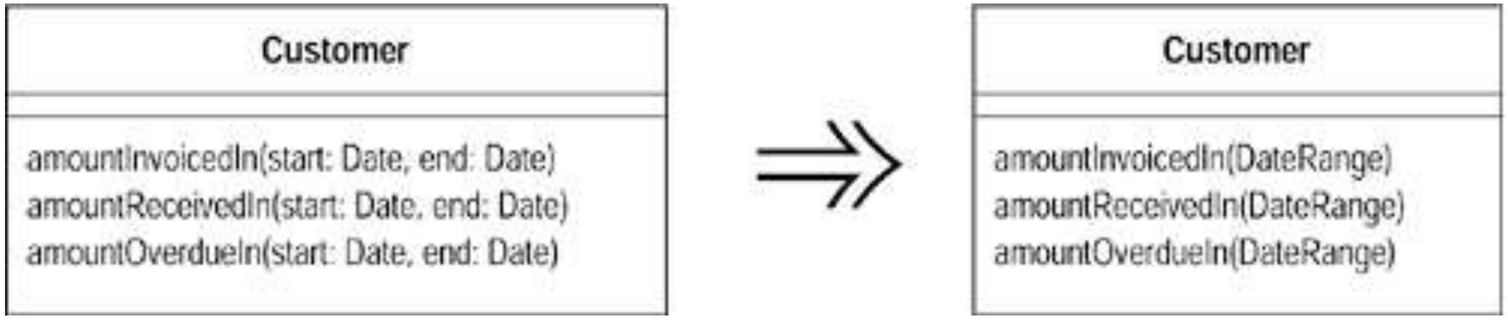
Replace Temp with Query

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

```
if(getBasePrice() > 1000) {
return getBasePrice() * 0.95;
}
else {
return getBasePrice() * 0.98;
}
double getBasePrice() {
return _quanitiy *
_itemPrice;
}
```

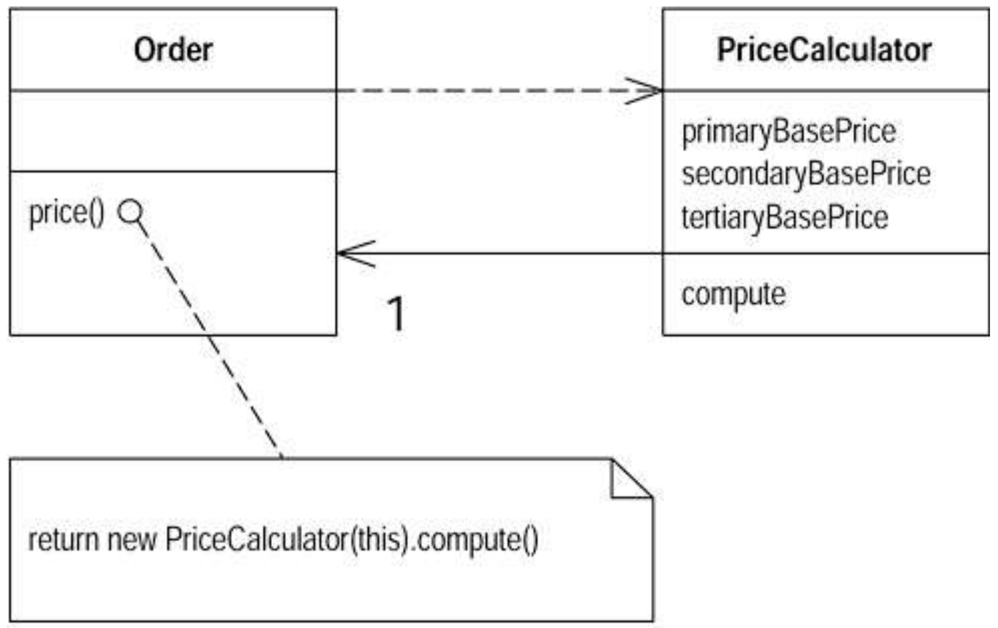
Remedy of Long method

Introduce Parameter Object


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

Replace Method with Method Object

```
//class Order...
double price() {
    double primaryBasePrice;
    double secondaryBasePrice;
    double tertiaryBasePrice;
// long computation;
...
}
```



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);
```

Lazy Class

- A class that isn't doing enough to carry its weight. We let the class die with dignity

Remedies

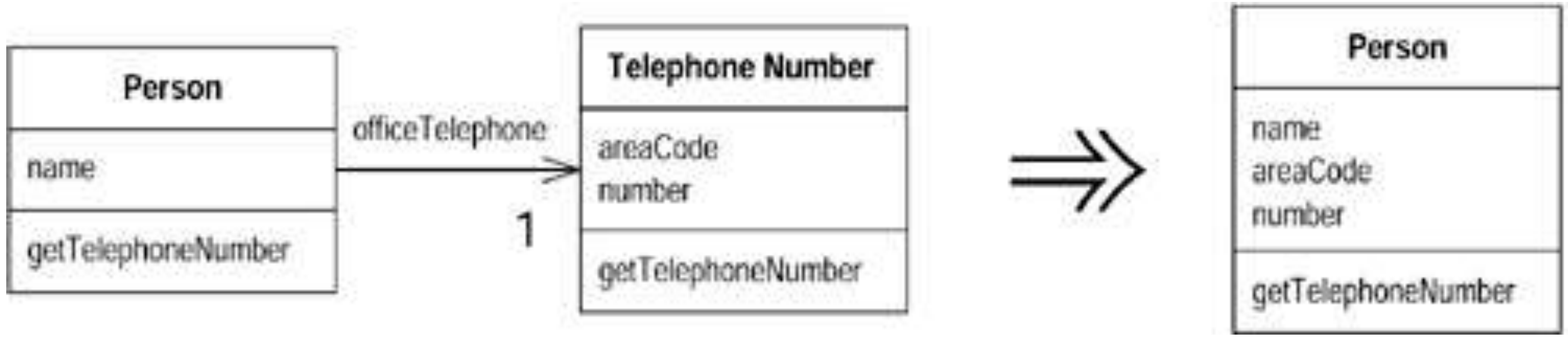
- ❑ Inline Class
- ❑ Collapse Hierarchy

Lazy Class (Cont'd)

```
public class Letter {
        private final String content;

        public Letter(String content) {
                this.content = content;
        }

        public String getContent() {
        return content;
        }
}
```

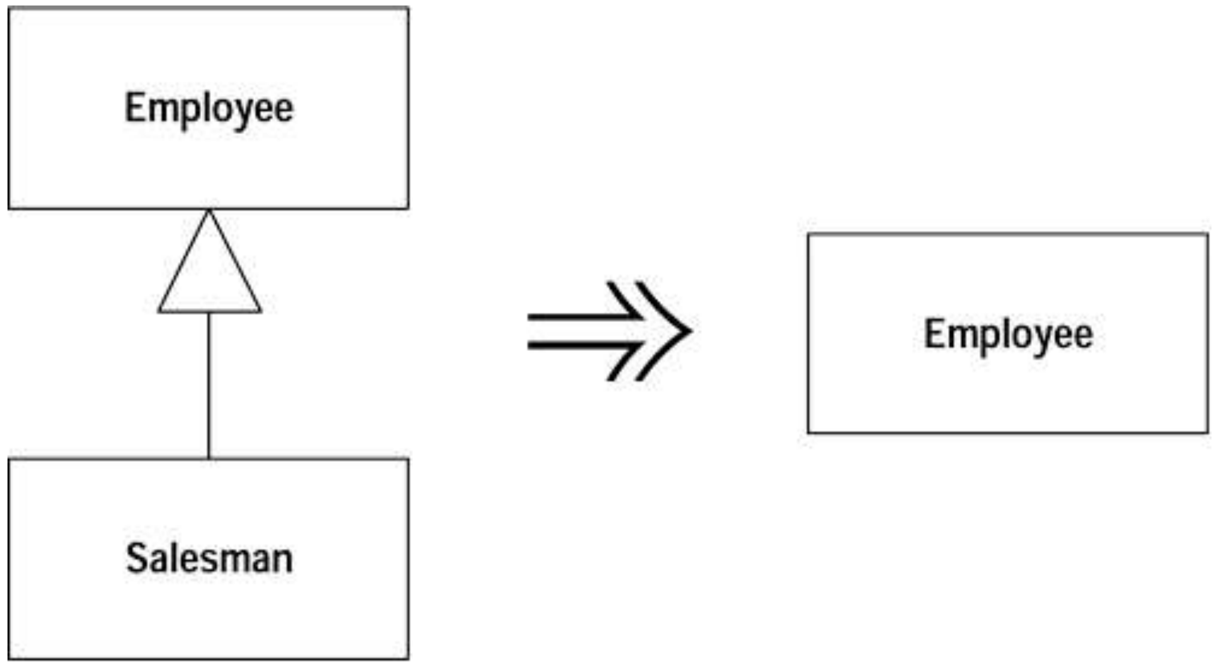
Remedy of Lazy Class

Inline Class



Remedy of Lazy Class

Collapse Hierarchy



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.

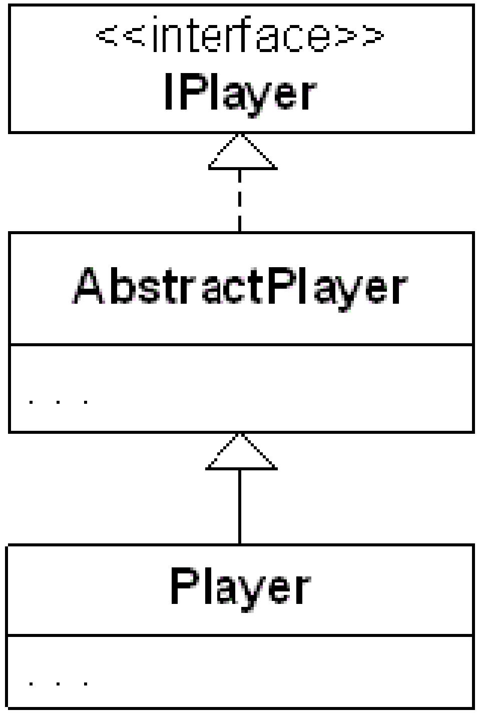
Remedy

- ❑ Collapse Hierarchy
- ❑ Inline Class
- ❑ Remove Parameter
- ❑ Rename Method

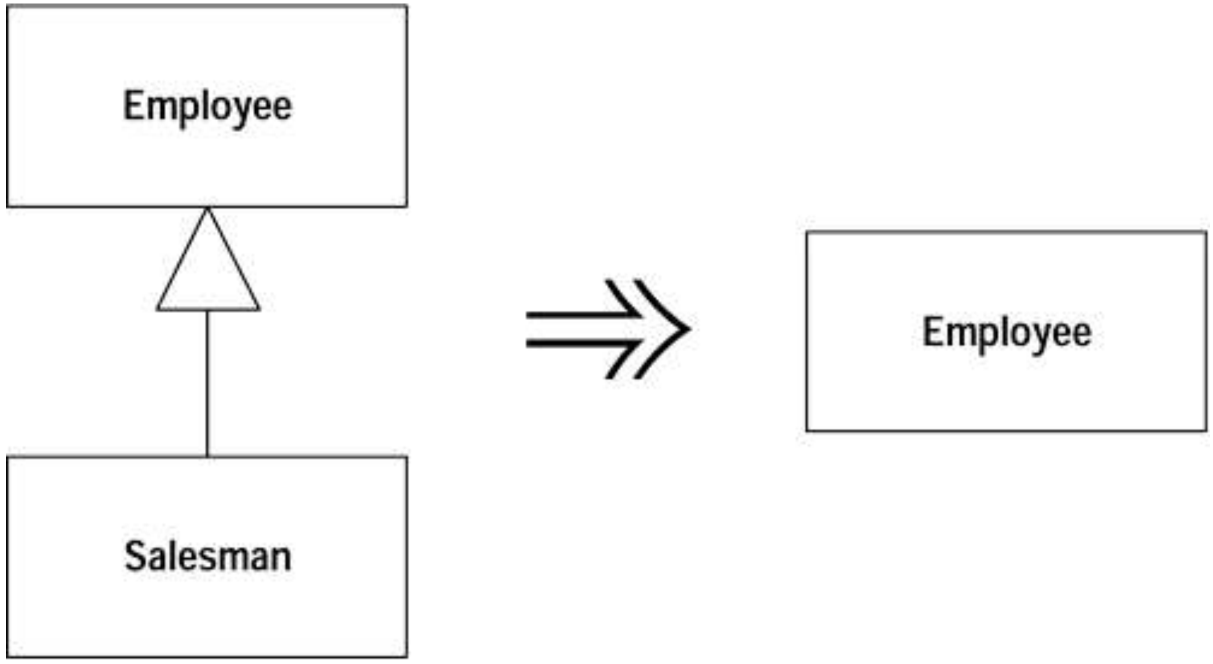
Speculative Generality (Cont'd)

```
public class Customer {
        private String name;
        private String address;
        private String salutation;
        private String otherDetails;
        private MailingSystem mailingSystem;

  public Customer(String name, String salutation, String add, String details, MailingSystem mSys) {
        this.name = name;
        this.address = add;
        this.salutation = salutation;
        this.otherDetails = details;
        this.mailingSystem = mSys;
 }

  String getName() {
        return name;
  }

  MailingSystem getMailingSystem() {
        return mailingSystem;
   }
}
```

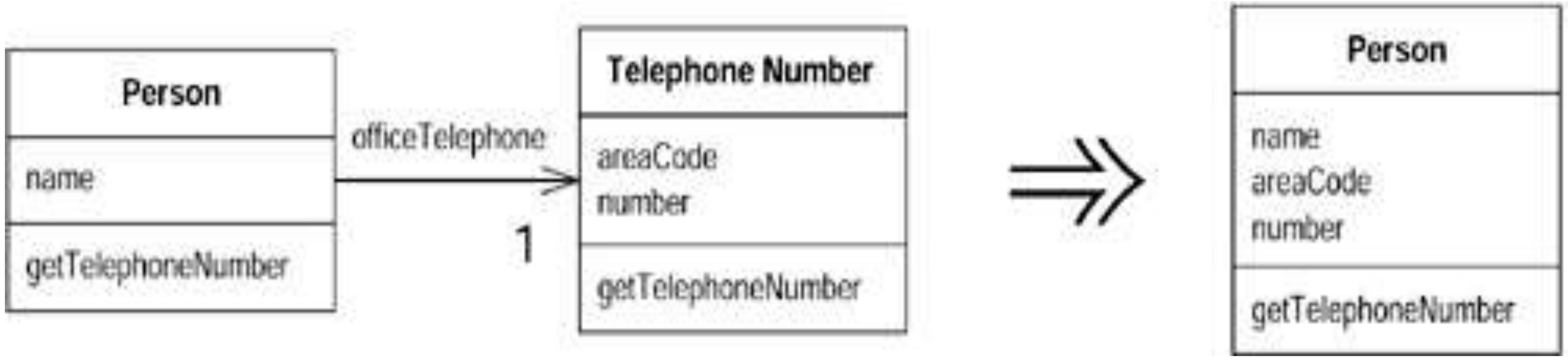
Speculative Generality (Cont’d)



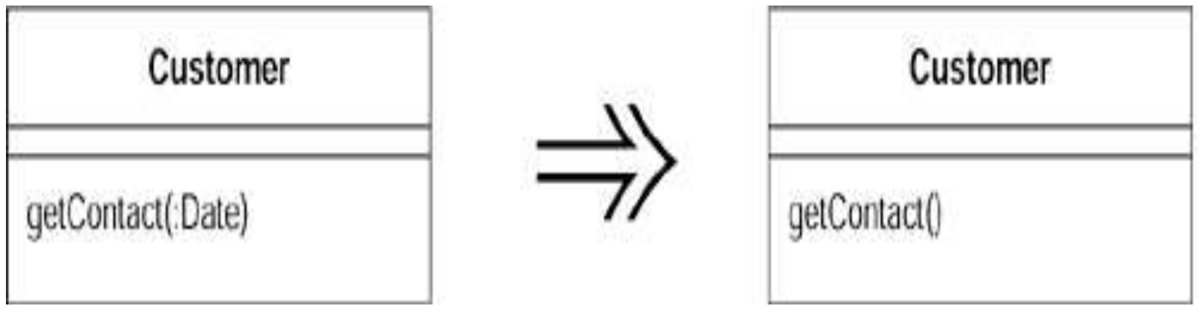
Collapse Hierarchy



Inline Class



Remove Parameter



Dead Code

- Code that is no longer used in a system or related system is Dead Code.
- Increased Complexity.
- Accidental Changes.
- More Dead Code

- Remedy

Dead Code (Cont’d)

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

```
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);
}
...
}
```

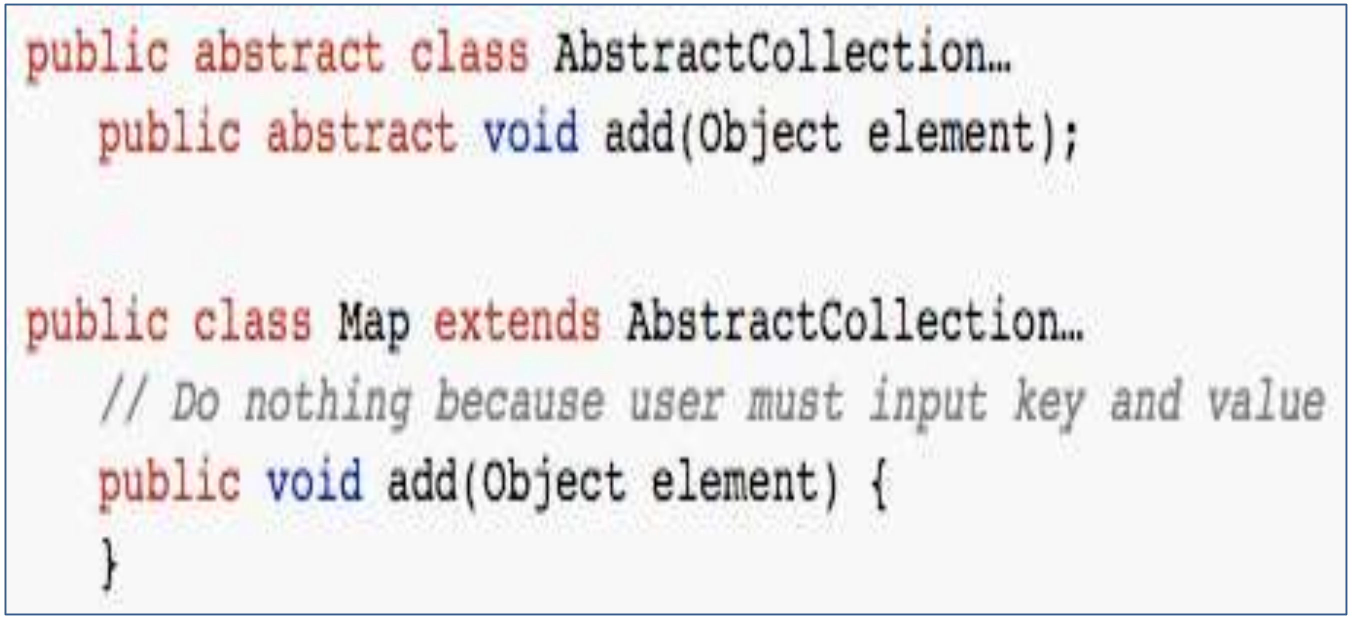
Refused Bequest

- This rather potent odor results when subclasses inherit code that they don’t want. In some cases, a subclass may “refuse the bequest” by providing a do nothing implementation of an inherited method.

Remedy

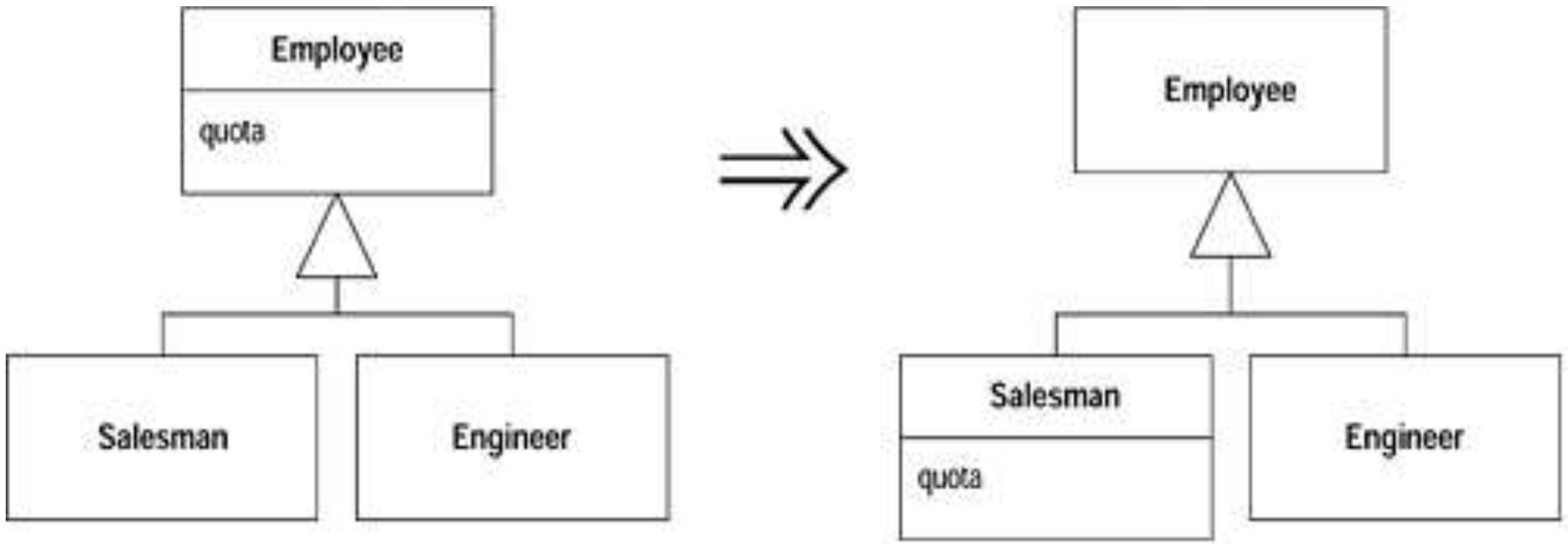
- ❑ Push Down Field/Method
- ❑ Replace Inheritance with Delegation

Refused Bequest (Cont’d)



```
public abstract class AbstractCollection…
   public abstract void add(Object element);

public class Map extends AbstractCollection…
   // Do nothing because user must input key and value
   public void add(Object element) {
   }
```

Push Down Method



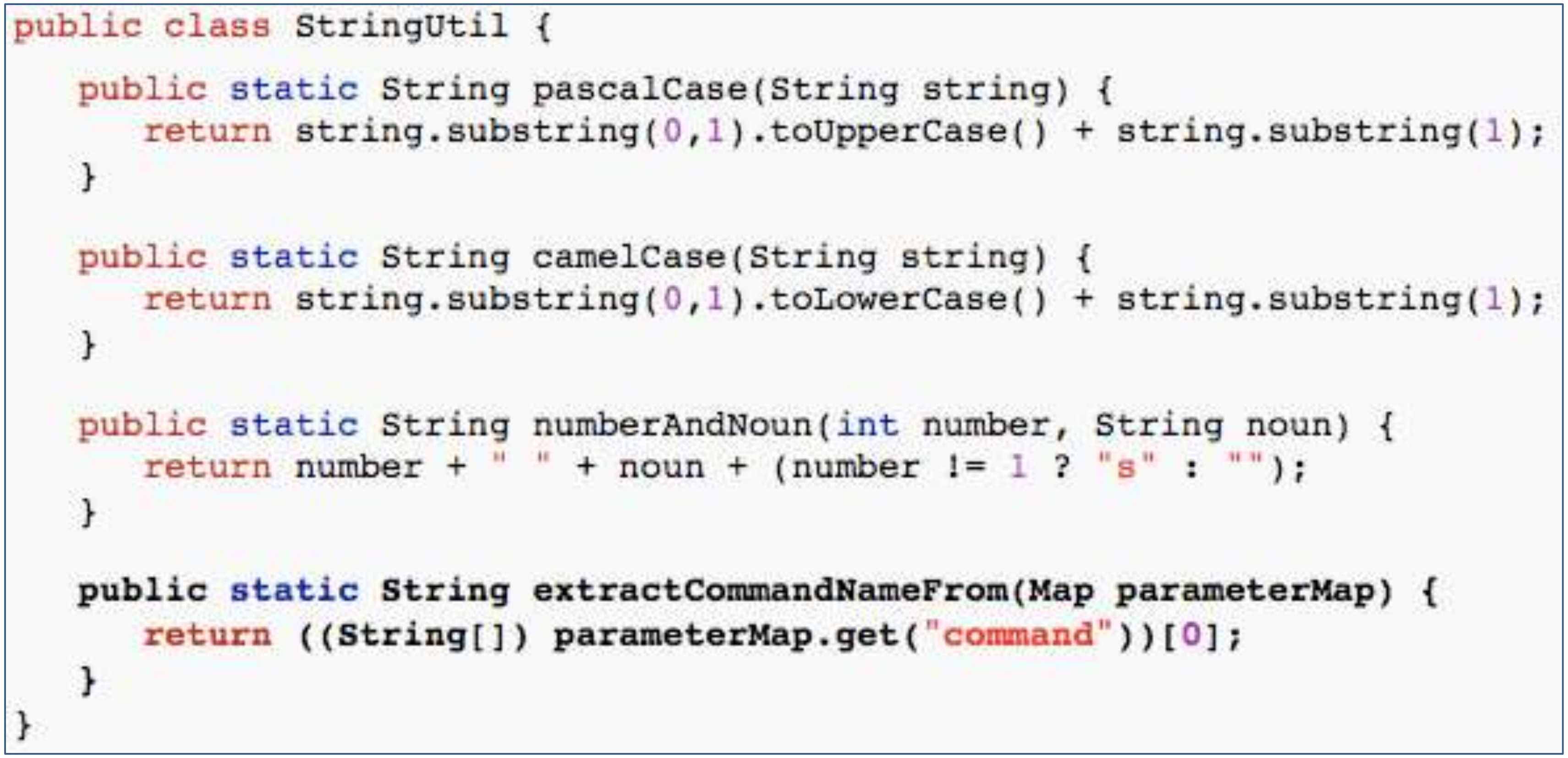
Black Sheep

- 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.

Remedy

- ❑ Move Method
- ❑ Extract Class

Black Sheep (Cont’d)



```java
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];
    }
}
```

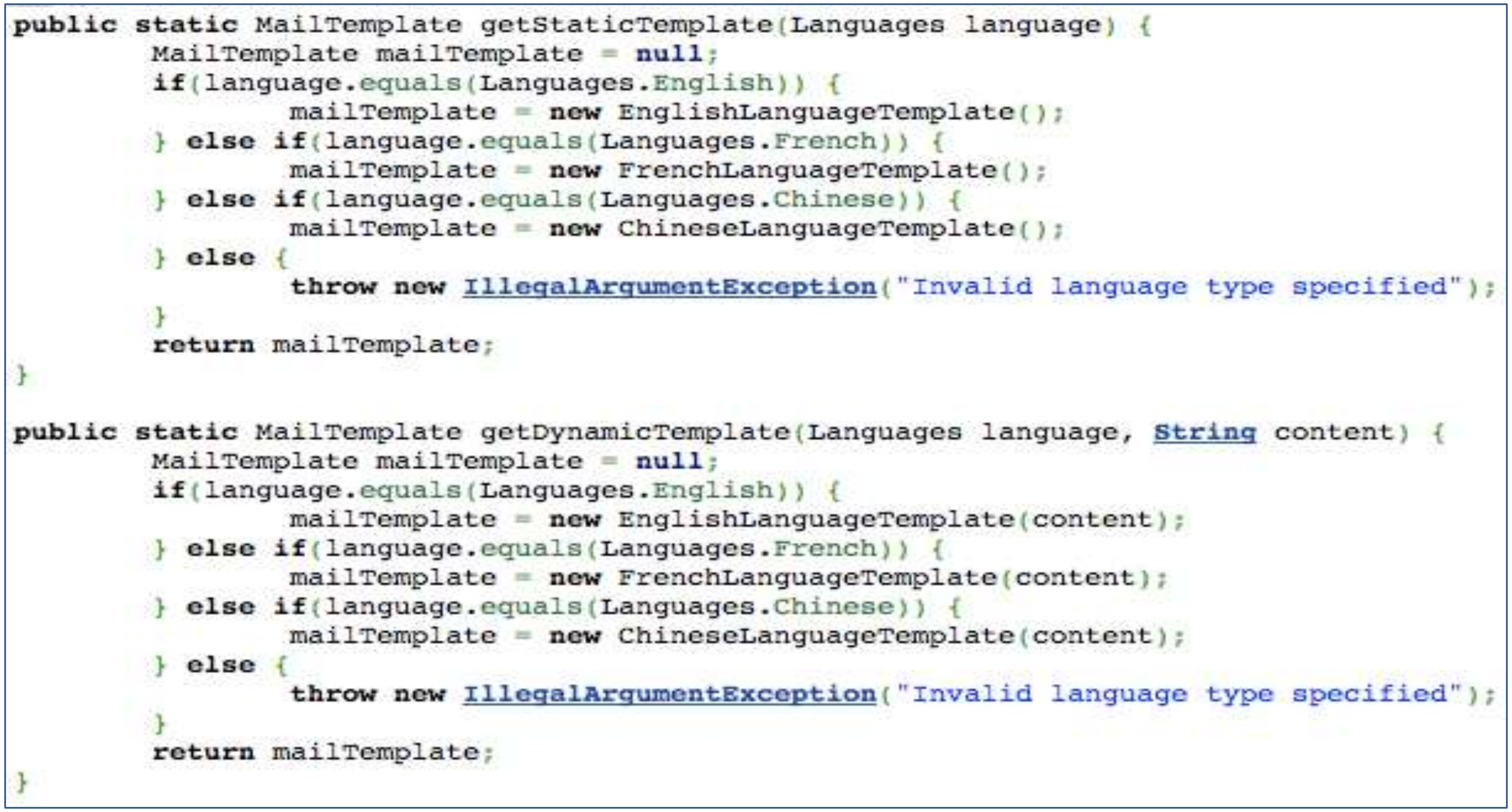
Duplicate Code

- 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

- Remedy
  - ❑ Extract Method
  - ❑ Pull Up Field
  - ❑ Form Template Method
  - ❑ Substitute Algorithm

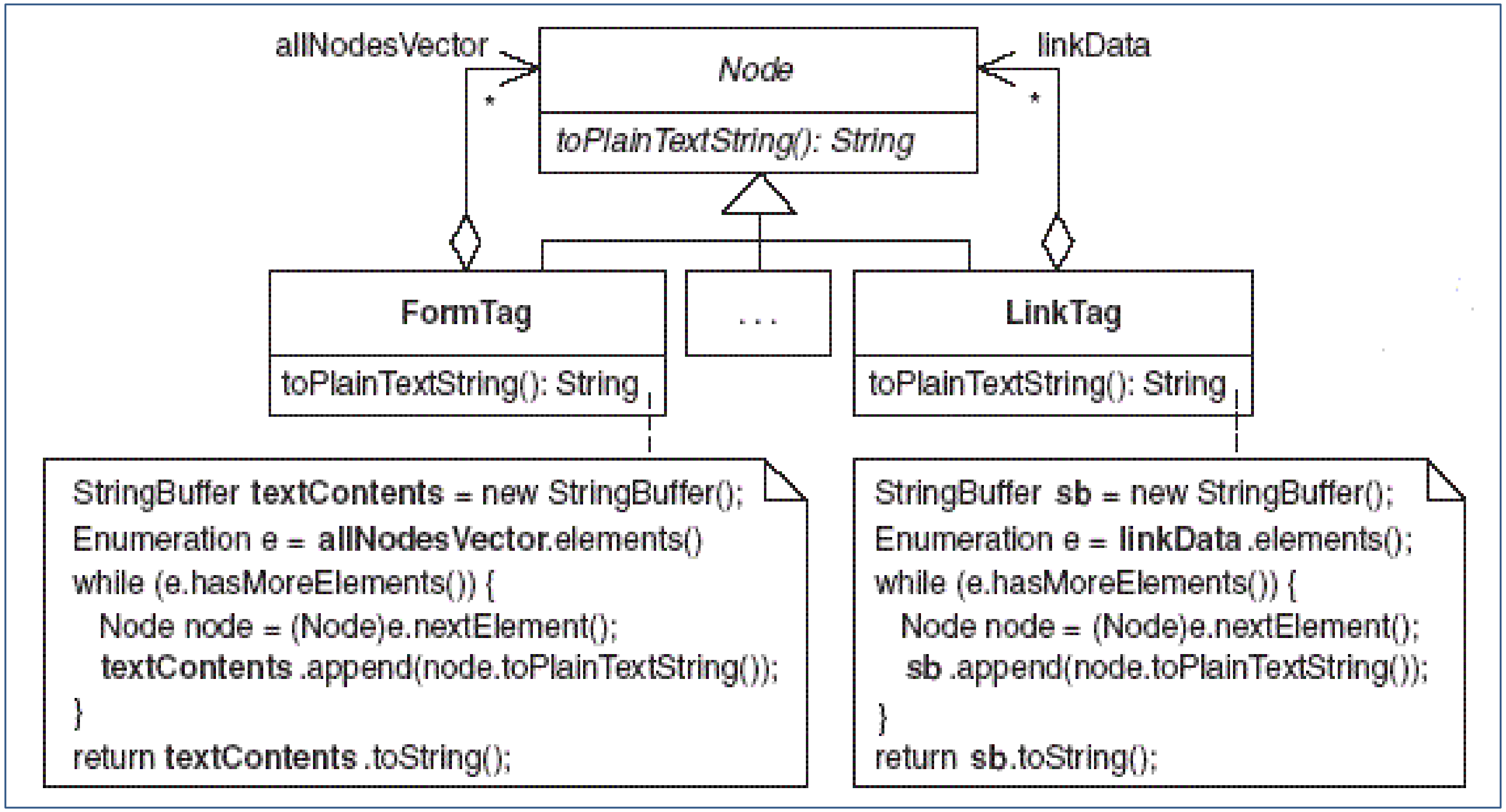
Duplicate Code (Cont’d)

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;
}
```

Duplicate Code (Cont’d)



allNodesVector
*
Node
toPlainTextString(): String
linkData
*
FormTag
toPlainTextString(): String
. . .
LinkTag
toPlainTextString(): String

```
StringBuffer textContents = new StringBuffer();
Enumeration e = allNodesVector.elements()
while (e.hasMoreElements()) {
   Node node = (Node)e.nextElement();
   textContents .append(node.toPlainTextString());
}
return textContents .toString();
```

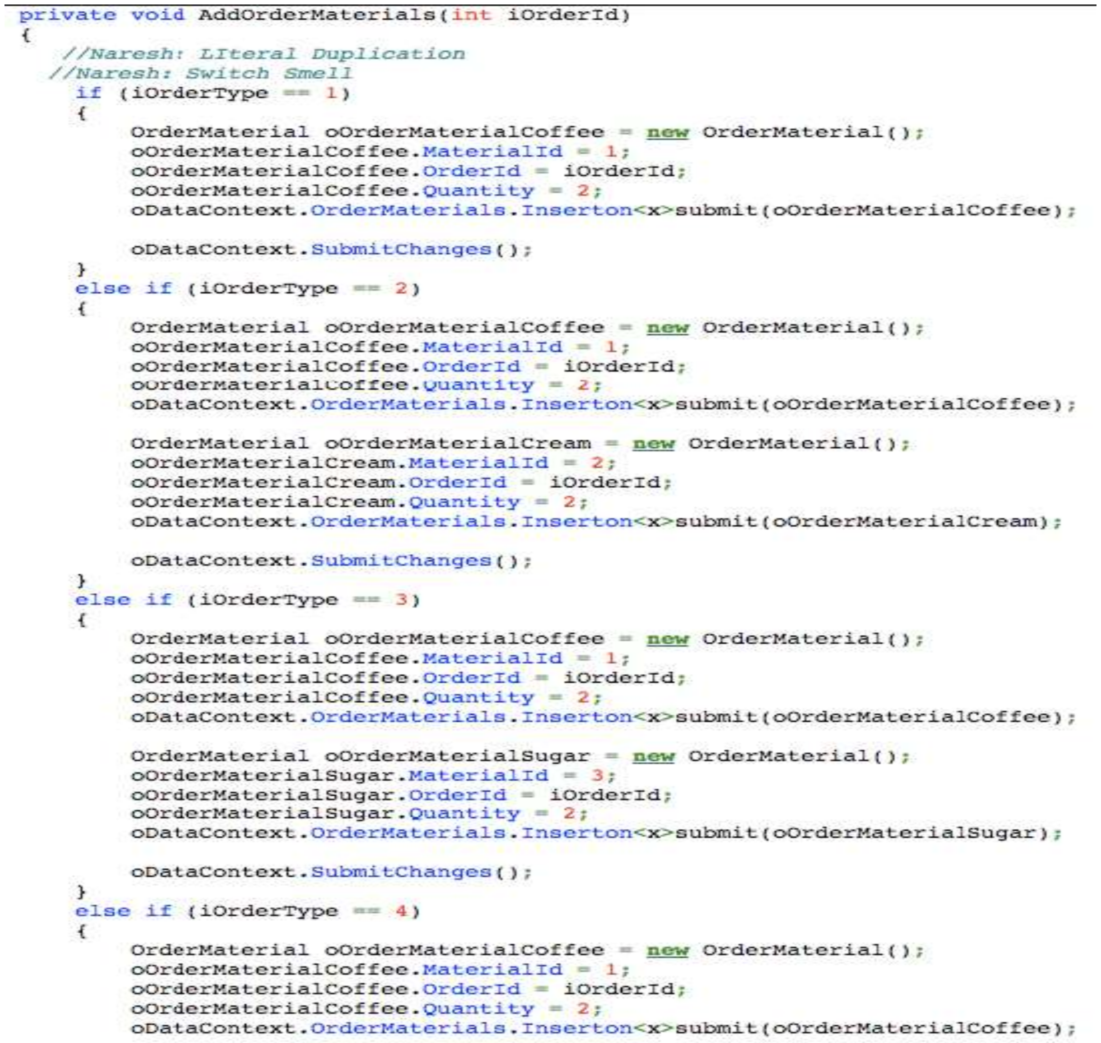
```
StringBuffer sb = new StringBuffer();
Enumeration e = linkData .elements();
while (e.hasMoreElements()) {
   Node node = (Node)e.nextElement();
   sb .append(node.toPlainTextString());
}
return sb.toString();
```

Duplicate Code (Cont'd)

```
public int addCustomer( int userId, Customer newCustomer) {
        List<Customer> customerList = customers.get(userId);
        int customerId = (int) Math.random() * 1000;
        // TODO: Logic to find/generate customer id.
        newCustomer.setId(customerId);
        if (customerList == null) {
                customerList = new LinkedList<Customer>();
                customers.put(userId, customerList);
        }
        customerList.add(newCustomer);
        return customerId;
}
```

```
public int addTemplate( int userId, Template newTemplate) {
        List<Template> templateList = templates.get(userId);
        int templateId = (int) Math.random() * 1000;
        // TODO: Logic to find/generate template id.
        newTemplate.setId(templateId);
        if (templateList == null) {
                templateList = new LinkedList<Template>();
                templates.put(userId, templateList);
        }
        templateList.add(newTemplate);
        return templateId;
}
```

Duplicate Code (Cont'd)

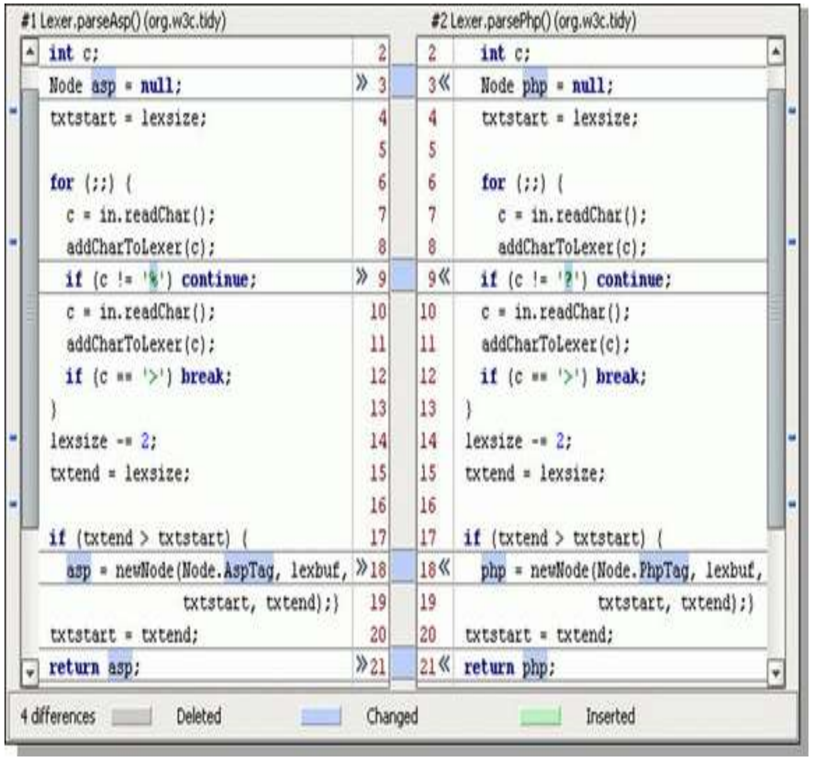


```
private void AddOrderMaterials(int iOrderId)
{
   //Naresh: LIteral Duplication
  //Naresh: Switch Smell
    if (iOrderType == 1)
    {
        OrderMaterial oOrderMaterialCoffee = new OrderMaterial();
        oOrderMaterialCoffee.MaterialId = 1;
        oOrderMaterialCoffee.OrderId = iOrderId;
        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)
    {
        OrderMaterial oOrderMaterialCoffee = new OrderMaterial();
        oOrderMaterialCoffee.MaterialId = 1;
        oOrderMaterialCoffee.OrderId = iOrderId;
        oOrderMaterialCoffee.Quantity = 2;
        oDataContext.OrderMaterials.Inserton<x>submit(oOrderMaterialCoffee);
```

Duplicate Code (Cont’d)

Levels of Duplication

- Literal
- Semantic
- Data Duplication
- Conceptual



#1 Lexer.parseAsp() (org.w3c.tidy)

```
int c;
Node asp = null;
txtstart = lexsize;

for (;;) {
  c = in.readChar();
  addCharToLexer(c);
  if (c != '%') continue;
  c = in.readChar();
  addCharToLexer(c);
  if (c == '>') break;
}
lexsize -= 2;
txtend = lexsize;

if (txtend > txtstart) {
  asp = newNode(Node.AspTag, lexbuf,
                txtstart, txtend);}
txtstart = txtend;
return asp;
```

#2 Lexer.parsePhp() (org.w3c.tidy)

```
int c;
Node php = null;
txtstart = lexsize;

for (;;) {
  c = in.readChar();
  addCharToLexer(c);
  if (c != '?') continue;
  c = in.readChar();
  addCharToLexer(c);
  if (c == '>') break;
}
lexsize -= 2;
txtend = lexsize;

if (txtend > txtstart) {
  php = newNode(Node.PhpTag, lexbuf,
                txtstart, txtend);}
txtstart = txtend;
return php;
```

4 differences ▨ Deleted 🟦 Changed 🟩 Inserted

Duplicate Code (Cont’d)

Literal Duplication

Same for loop in 2 places

Duplicate Code (Cont'd)

Semantic Duplication

1st Level - For and For Each Loop
2nd Level - Loop v/s Lines repeated
3rd Level - Loop v/s Recursion

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

v/s

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

Duplicate Code (Cont’d)

Data Duplication

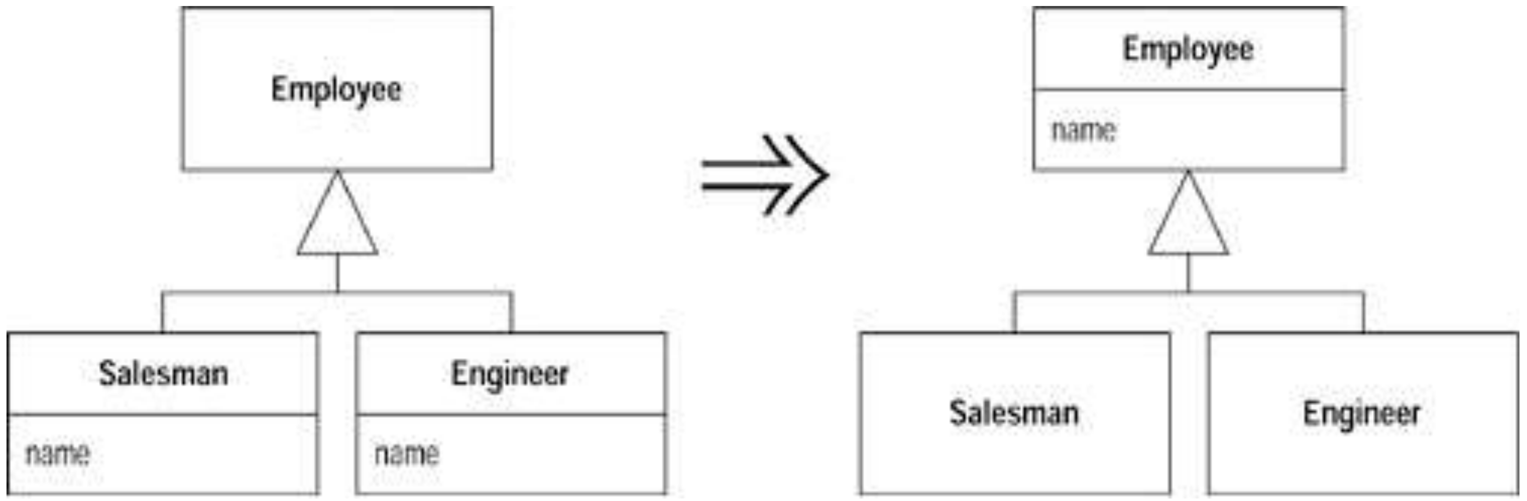
Some constant declared in 2 classes (test and production)

Duplicate Code (Cont'd)
Conceptual Duplication

Two Algorithms to Sort elements (Bubble sort and Quick sort)

Remedy of duplicate code

Pull Up Field



Remedy of duplicate code

Substitute Algorithm

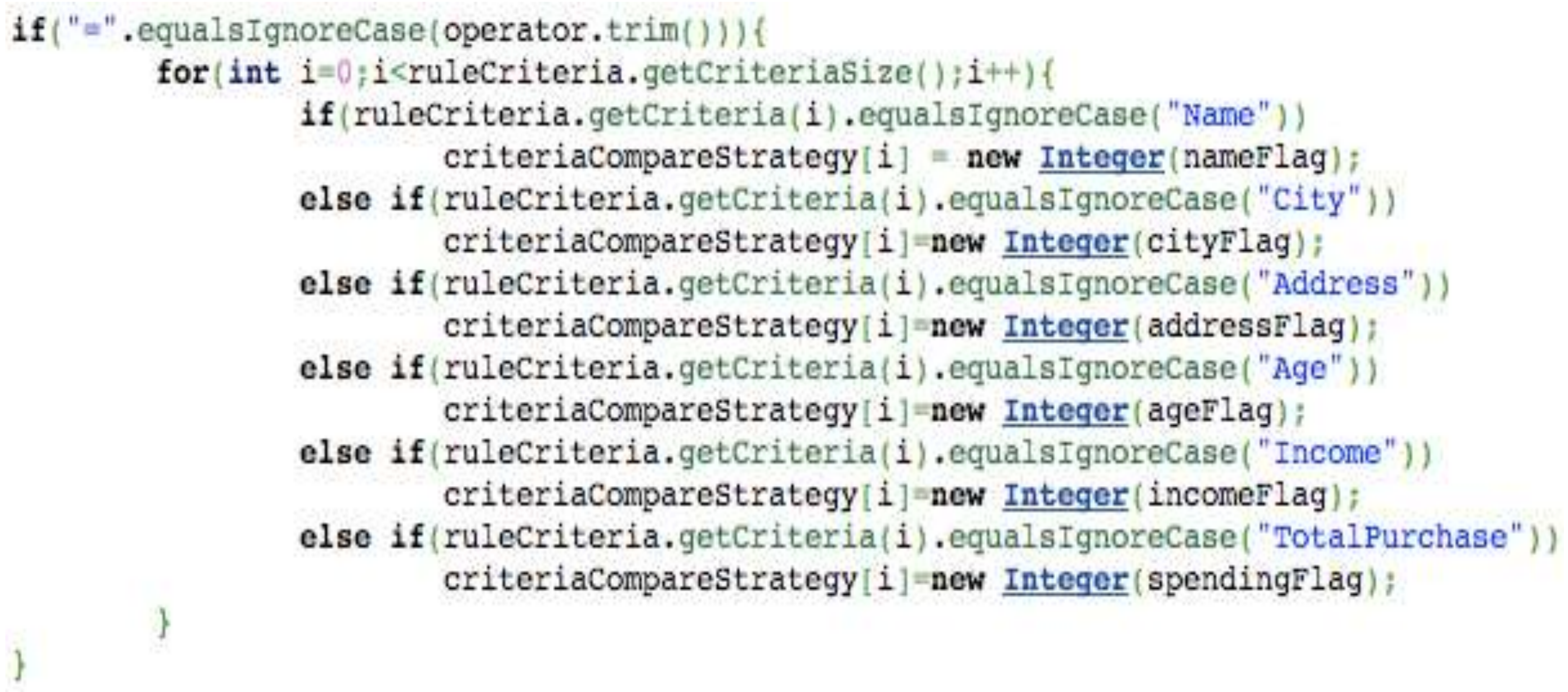
```
String foundPerson(String[] people){
for (int i = 0; i < people.length; i++) {
if (people[i].equals ("Don")){
return "Don";
}
if (people[i].equals ("John")){
return "John";
}
if (people[i].equals ("Kent")){
return "Kent";
}
}
return "";
}
```

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

Switch Statement

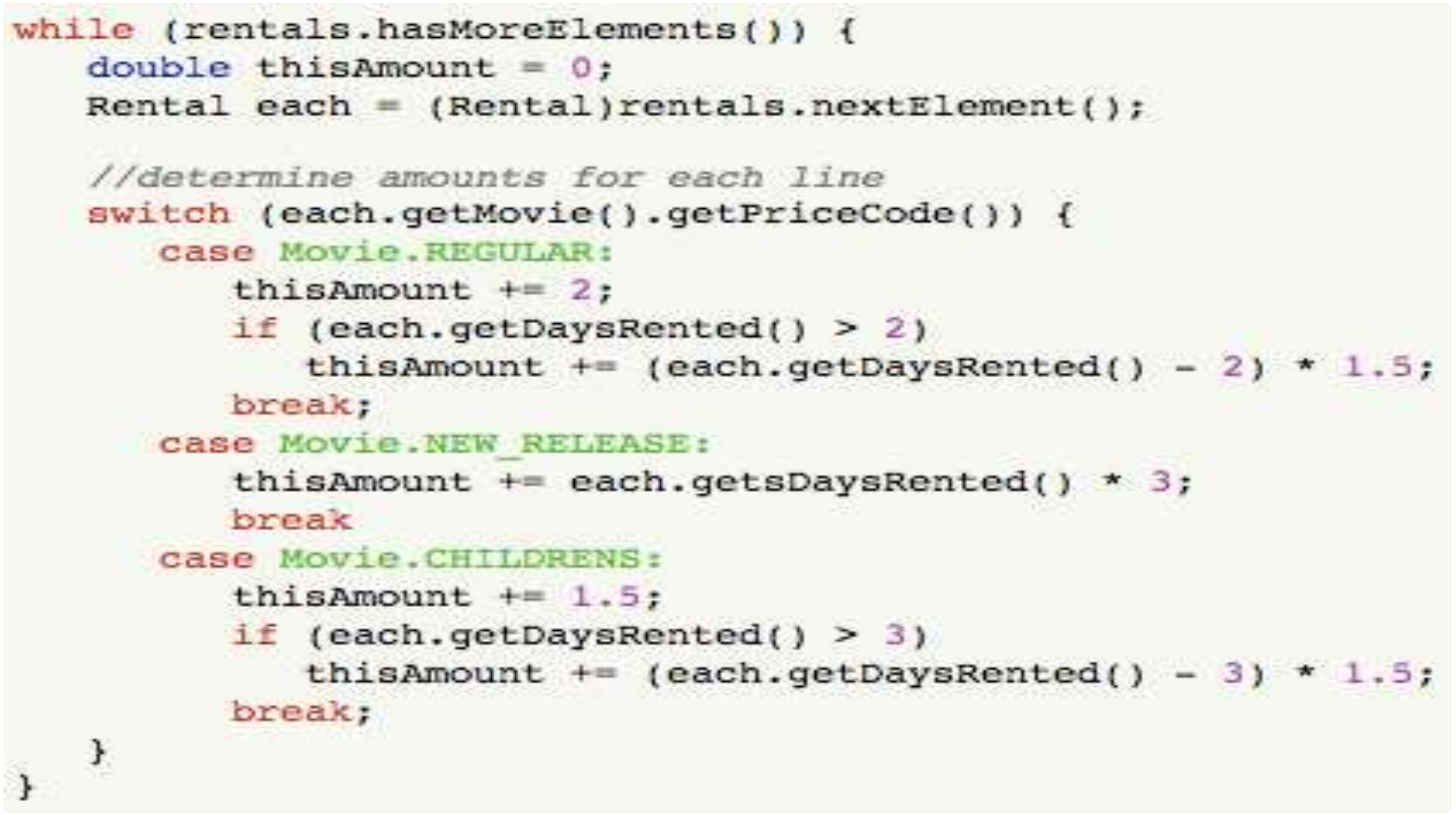
- This smell exists when the same switch statement (or “if…else if…else if”
- statement) is duplicated across a system.
- Such duplicated code reveals a lack of object-orientation and a missed
- opportunity to rely on the elegance of polymorphism.
- Remedy:
  - Replace Type Code with Polymorphism
  - Replace Type Code with State / Strategy
  - Replace Parameter with Explicit Methods
  - Introduce Null Object.

Switch Smell (Cont’d)



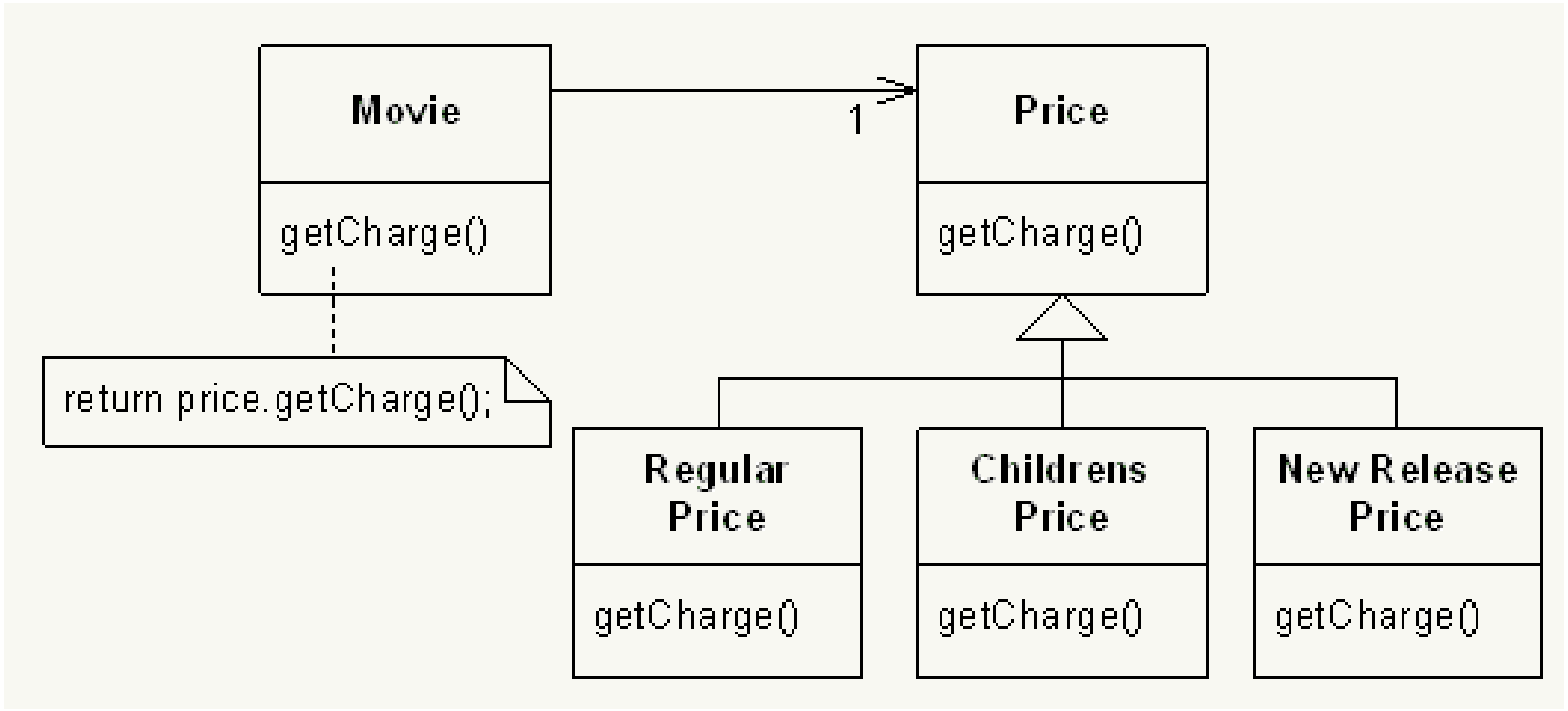
```
if("=".equalsIgnoreCase(operator.trim())){
        for(int i=0;i<ruleCriteria.getCriteriaSize();i++){
                if(ruleCriteria.getCriteria(i).equalsIgnoreCase("Name"))
                        criteriaCompareStrategy[i] = new Integer(nameFlag);
                else if(ruleCriteria.getCriteria(i).equalsIgnoreCase("City"))
                        criteriaCompareStrategy[i]=new Integer(cityFlag);
                else if(ruleCriteria.getCriteria(i).equalsIgnoreCase("Address"))
                        criteriaCompareStrategy[i]=new Integer(addressFlag);
                else if(ruleCriteria.getCriteria(i).equalsIgnoreCase("Age"))
                        criteriaCompareStrategy[i]=new Integer(ageFlag);
                else if(ruleCriteria.getCriteria(i).equalsIgnoreCase("Income"))
                        criteriaCompareStrategy[i]=new Integer(incomeFlag);
                else if(ruleCriteria.getCriteria(i).equalsIgnoreCase("TotalPurchase"))
                        criteriaCompareStrategy[i]=new Integer(spendingFlag);
        }
}
```

Switch Smell (Cont'd)



```
while (rentals.hasMoreElements()) {
    double thisAmount = 0;
    Rental each = (Rental)rentals.nextElement();

    //determine amounts for each line
    switch (each.getMovie().getPriceCode()) {
        case Movie.REGULAR:
            thisAmount += 2;
            if (each.getDaysRented() > 2)
                thisAmount += (each.getDaysRented() - 2) * 1.5;
            break;
        case Movie.NEW_RELEASE:
            thisAmount += each.getsDaysRented() * 3;
            break
        case Movie.CHILDRENS:
            thisAmount += 1.5;
            if (each.getDaysRented() > 3)
                thisAmount += (each.getDaysRented() - 3) * 1.5;
            break;
    }
}
```

Replace Type Code with Polymorphism



Replace Parameter with Method

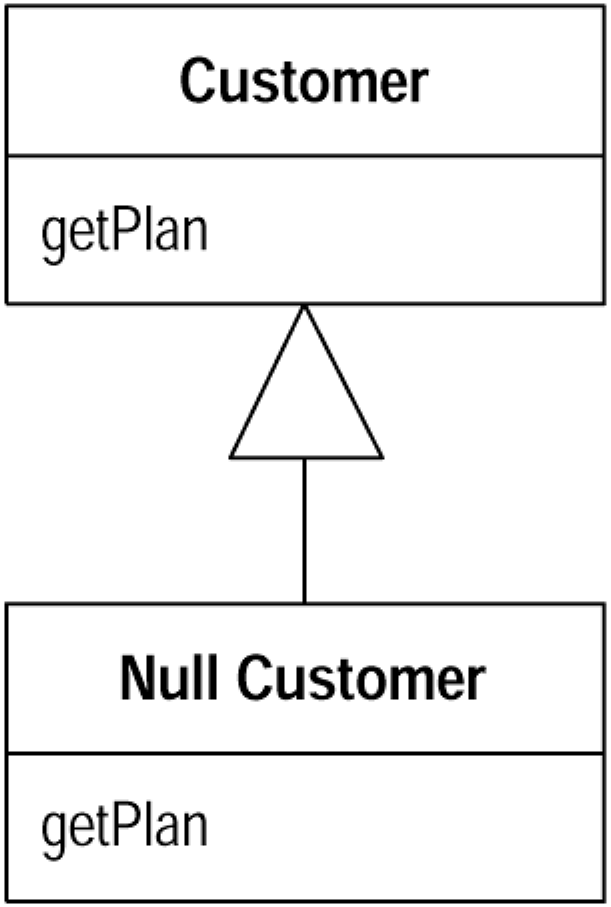
```
void setValue (String name, int value) {
if (name.equals("height"))
this.height = value;
if (name.equals("width"))
this.width = value;
}
```

```
void setHeight(int h) {
this.height = h;
}
void setWidth (int w) {
this.width = w;
}
```

Remedy of Switch Smell

Introduce Null Object

```
// In client class
Customer customer = site.getCustomer();
BillingPlan plan;
if (customer == null) plan = BillingPlan.basic();
else plan = customer.getPlan();

// In client class
Customer customer = site.getCustomer();
BillingPlan plan = customer.getPlan();

// In Null Customer
public BillingPlan getPlan(){
return BillingPlan.basic();
}
```

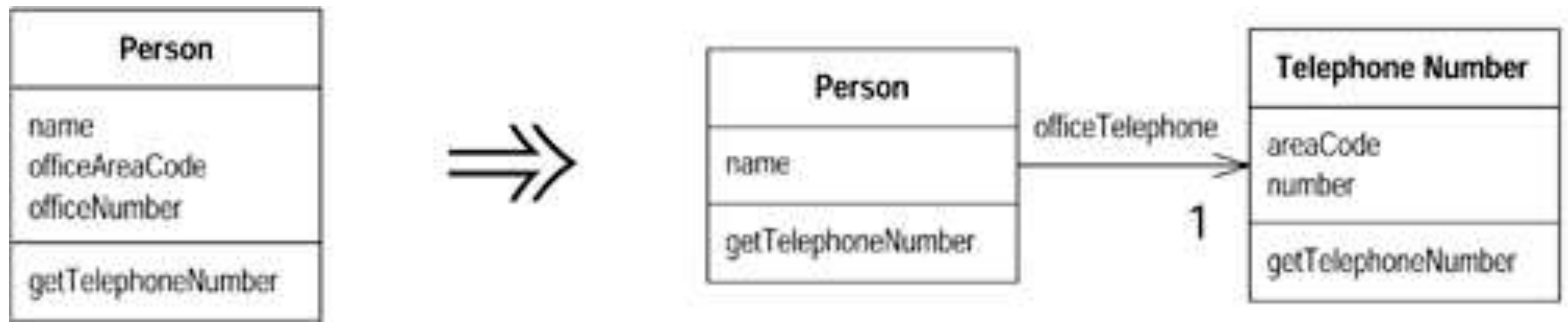


Large Class

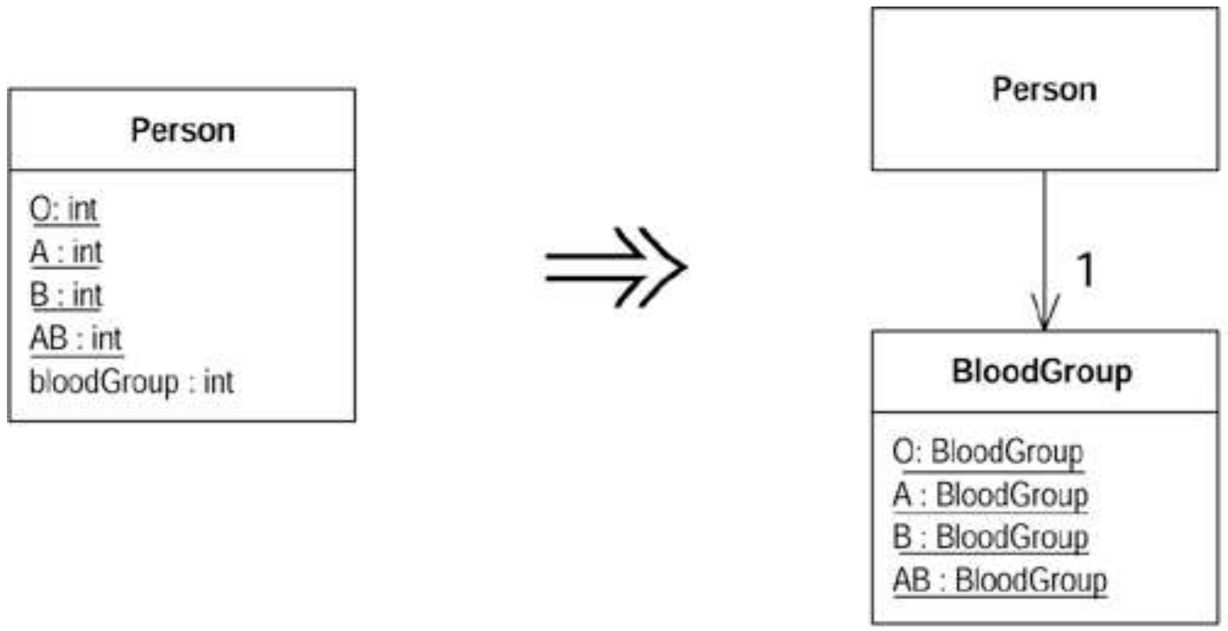
- Like people, classes suffer when they take on too many responsibilities.
- GOD Objects
- Fowler and Beck note that the presence of too many instance variables usually indicates that a class is trying to do too much. In general, large classes typically contain too many responsibilities.

- Remedies
  - Extract Class
  - Replace Type Code with Class/Subclass
  - Replace Type Code with State/Strategy
  - Replace Conditional with Polymorphism
  - Extract Interface
  - Duplicate Observed Data

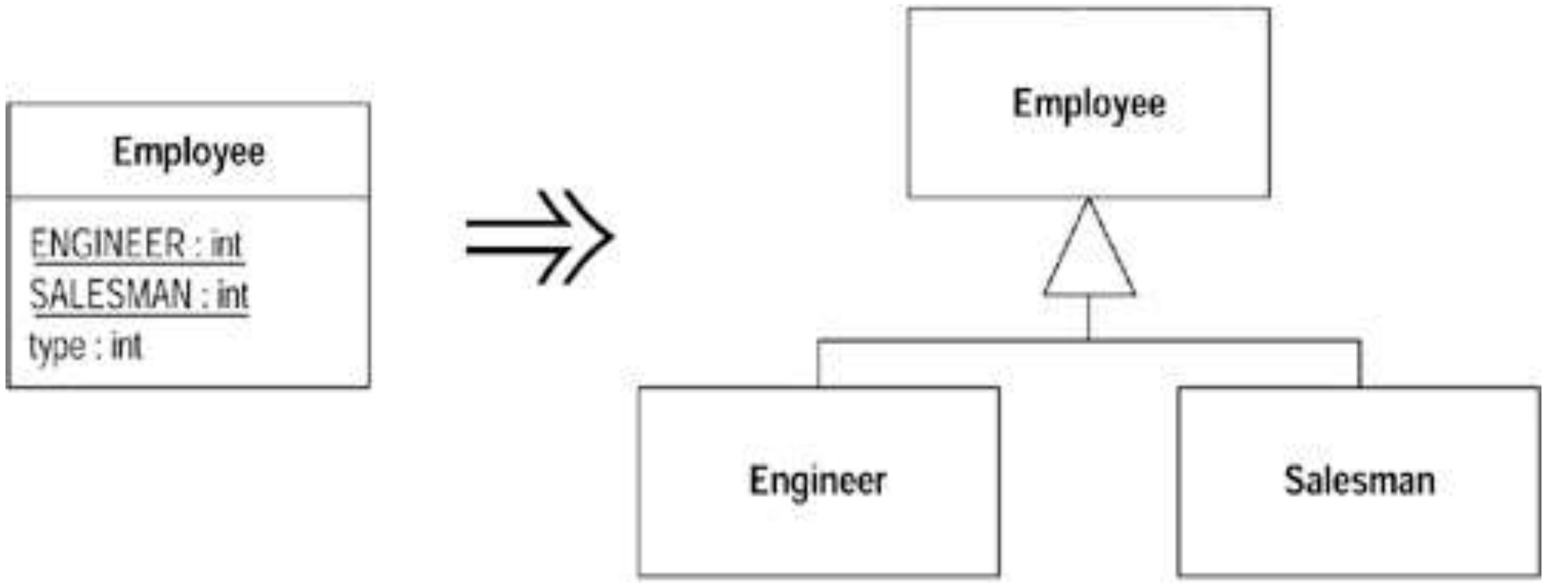
Extract Class



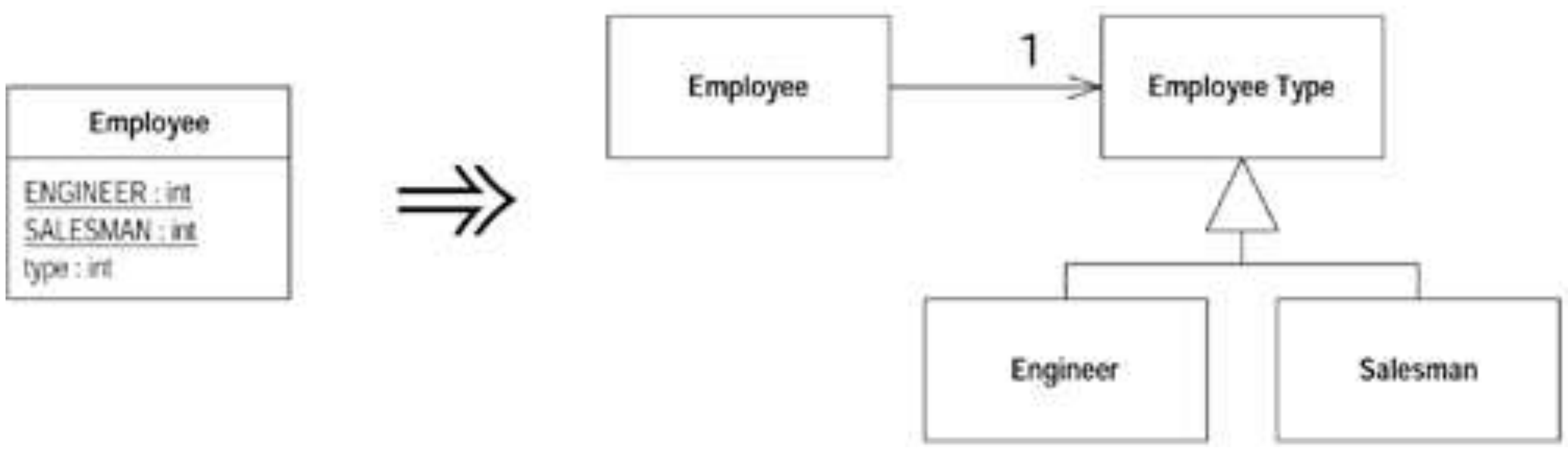
Replace Type Code with Class



Replace Type Code with Subclasses



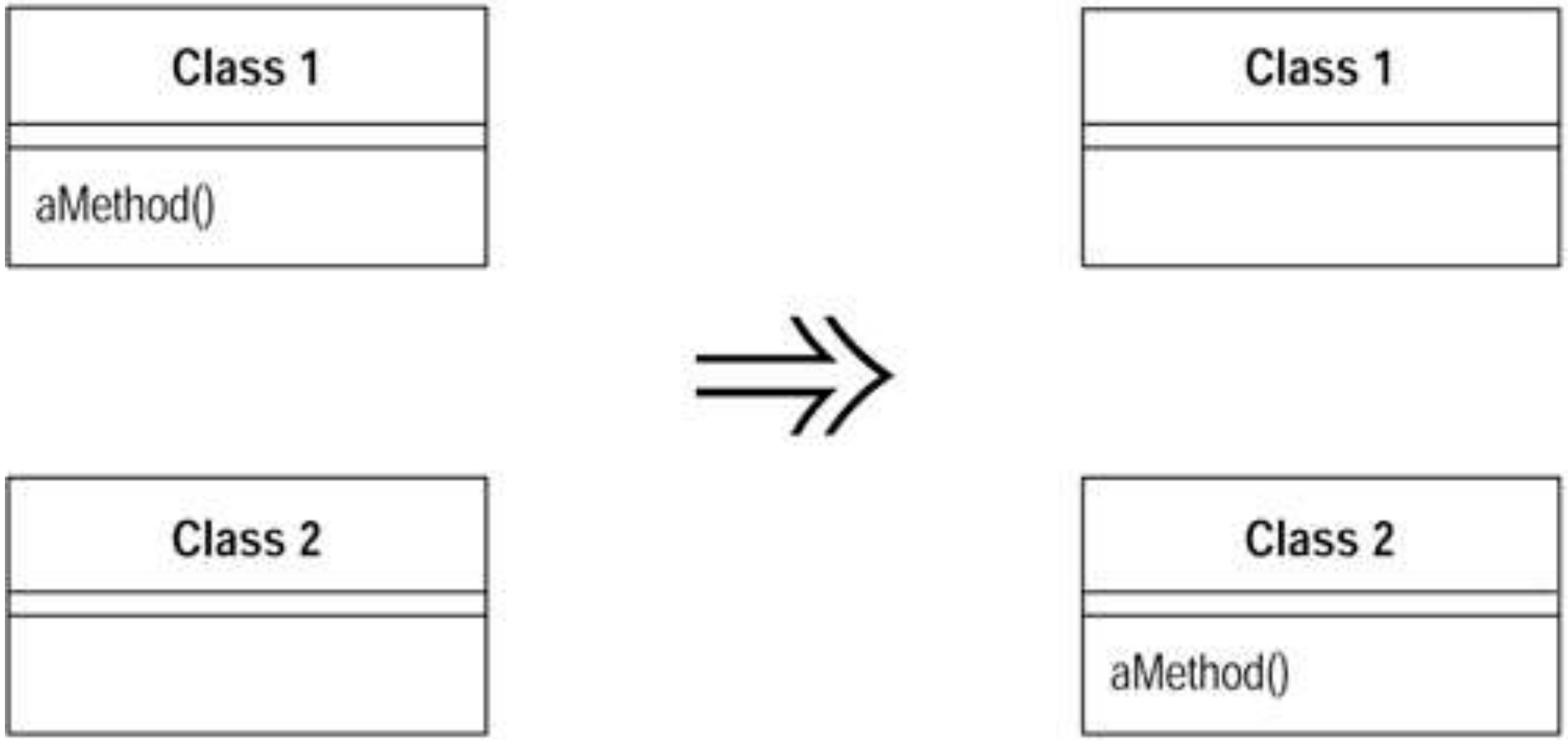
Replace Type Code with State/Strategy



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

Move Field



Contrived complexity

Forced usage of overly complicated design patterns where simpler design would suffice.

"Any intelligent fool can make things bigger, more complex, and more violent. It takes a touch of genius…and a lot of courage to move in the opposite direction." ~ E.F. Schumacher

Data clumps

- Whenever two or three values are gathered together in a class

Remedy: Extract class

Temporary fields

- Class has a variable which is only used in some situation.

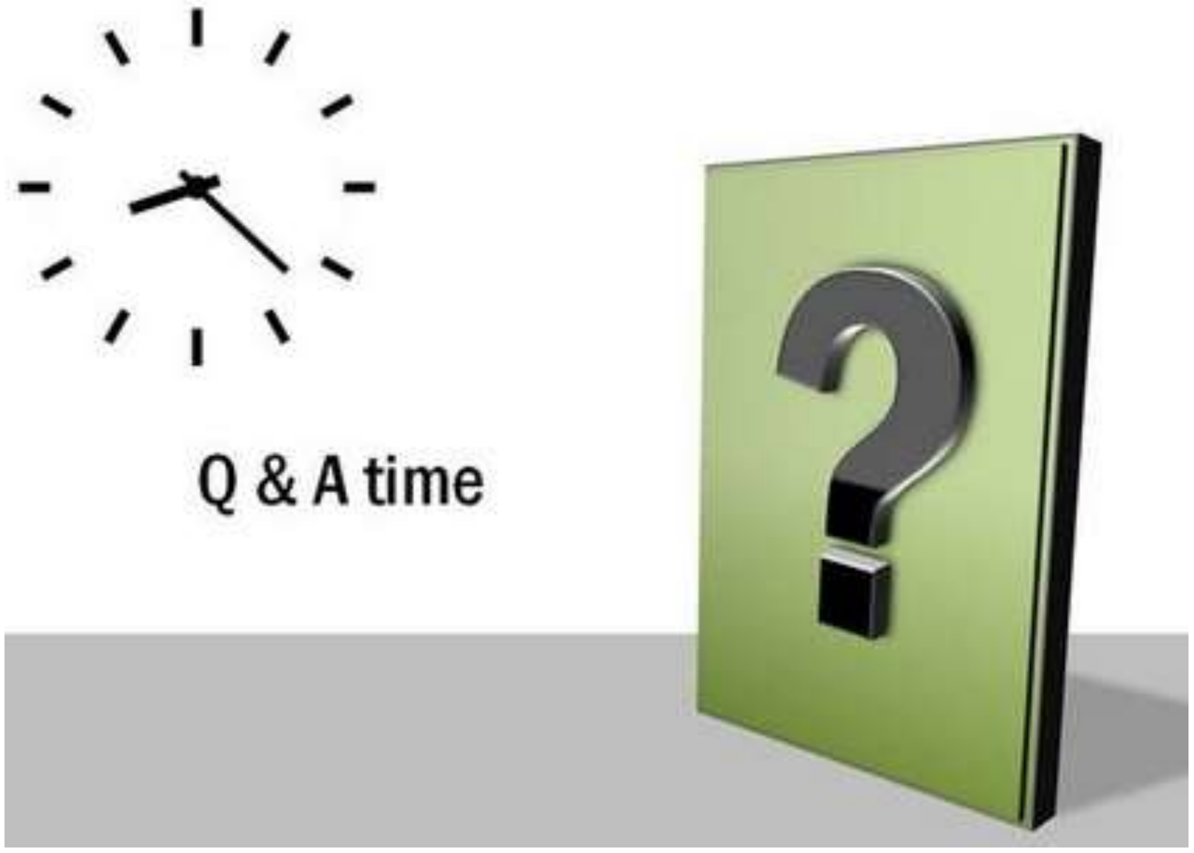
Remedy

- Move field

Acknowledgements

- Martin Fowler
- Kent Beck
- Naresh Jain
- Ron Jeffries
- Robert C. Martin
- Industrial Logic
- Though Works
- Nerd Castle
- The Other Resources

THANK
YOU


Q & A time