CSCB07 - Software Design

Code Smells and Refactoring

Fall 2023

What is a code smell?

- A code smell is a structure in the code that indicates a potentially deeper problem.

- In many cases, a code smell (and the underlying problem) can be addressed by refactoring

Fowler, Martin. Refactoring: Improving the Design of Existing Code. Boston, MA, USA: Addison-Wesley, 1999.

Examples of code smells

- Duplicated code
- Long method
- Large class
- Long parameter list
- Excessive commenting
- Feature envy
    - Occurs when a method seems more interested in a class other than the one it is in
- Data clumps
    - Occurs when the same data items are used together in a lot of places
- Refused bequest
    - Occurs when a subclass inherits data/method(s) that it does not need

Fowler, Martin. Refactoring: Improving the Design of Existing Code. Boston, MA, USA: Addison-Wesley, 1999.

What is 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*
- Code quality deteriorates with time, even for a well-designed system
- Advantages of refactoring:
  - ➢ Improving software design
  - ➢ Making software easier to understand
  - ➢ Finding bugs more easily
  - ➢ Speeding up development

Fowler, Martin. Refactoring: Improving the Design of Existing Code. Boston, MA, USA: Addison-Wesley, 1999.

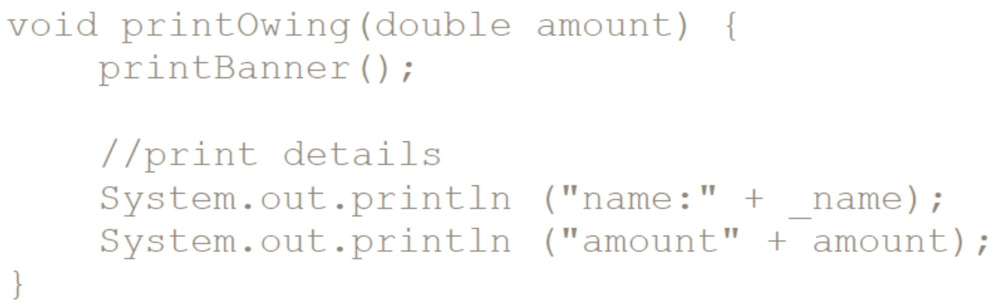
How to Refactor?

- Apply changes in a sequence of small steps
- Do not add any new functionality
- Make sure that all existing tests pass and add new ones if need be

Fowler, Martin. Refactoring: Improving the Design of Existing Code. Boston, MA, USA: Addison-Wesley, 1999.

Extract Method

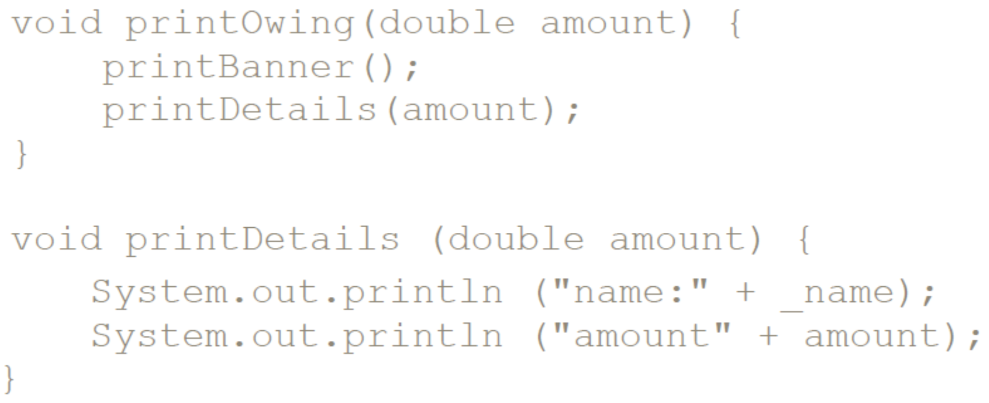
- **Issue:** You have a code fragment that can be grouped together.
- **Refactoring:** Turn the fragment into a method whose name explains the purpose of the method.
- **Example**



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

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

⇒



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

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

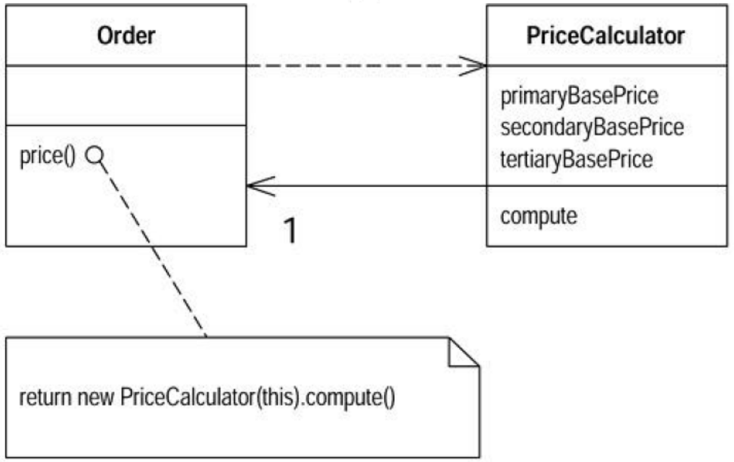
Fowler, Martin. Refactoring: Improving the Design of Existing Code. Boston, MA, USA: Addison-Wesley, 1999.

Replace Method with Method Object

- **Issue:** You have a long method that uses local variables in such a way that you cannot apply Extract Method.
- **Refactoring:** Turn the method into its own object so that all the local variables become fields on that object.
- **Example**

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

⇒

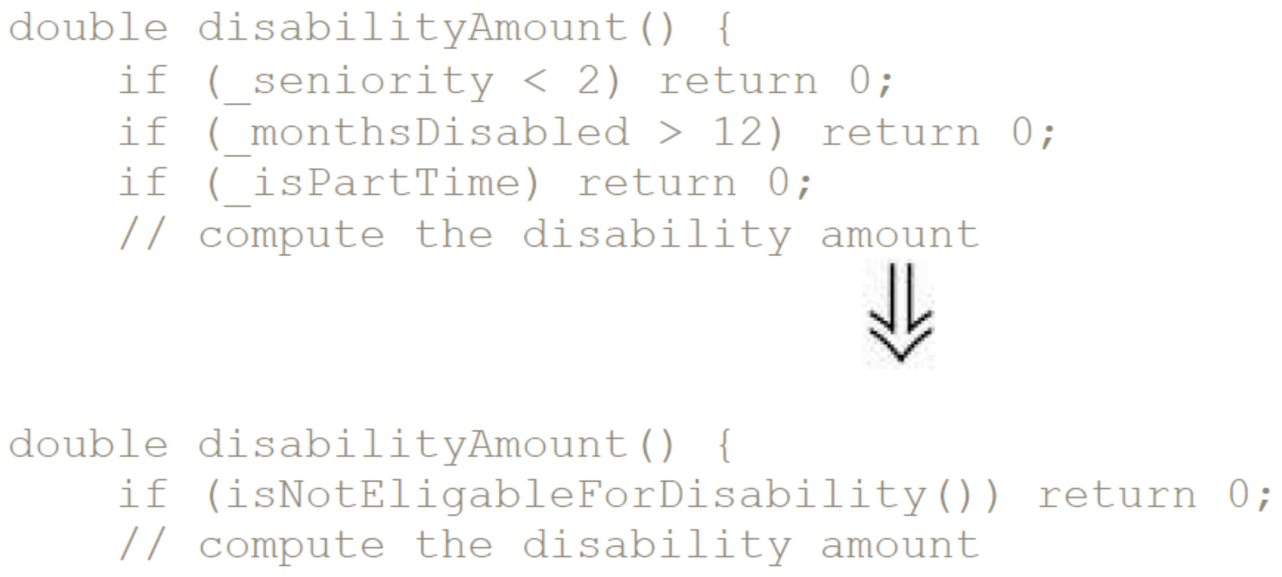


```
return new PriceCalculator(this).compute()
```

Fowler, Martin. Refactoring: Improving the Design of Existing Code. Boston, MA, USA: Addison-Wesley, 1999.

Consolidate Conditional Expression

- **Issue:** You have a sequence of conditional tests with the same result
- **Refactoring:** Combine into a single conditional expression and extract it.
- **Example**



```
double disabilityAmount() {
    if (_seniority < 2) return 0;
    if (_monthsDisabled > 12) return 0;
    if (_isPartTime) return 0;
    // compute the disability amount
```

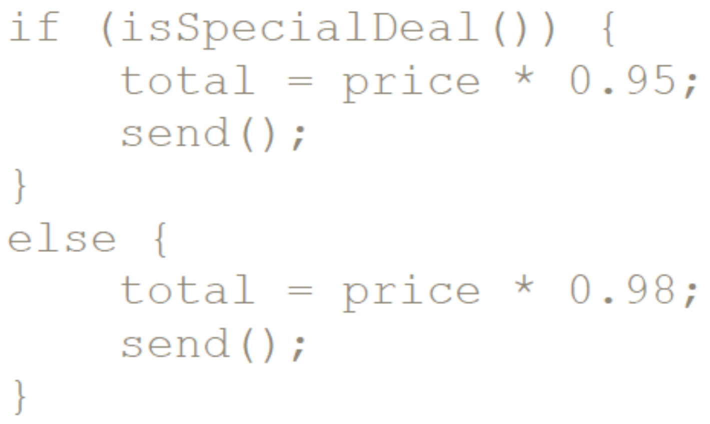
⇓

```
double disabilityAmount() {
    if (isNotEligableForDisability()) return 0;
    // compute the disability amount
```

Fowler, Martin. Refactoring: Improving the Design of Existing Code. Boston, MA, USA: Addison-Wesley, 1999.

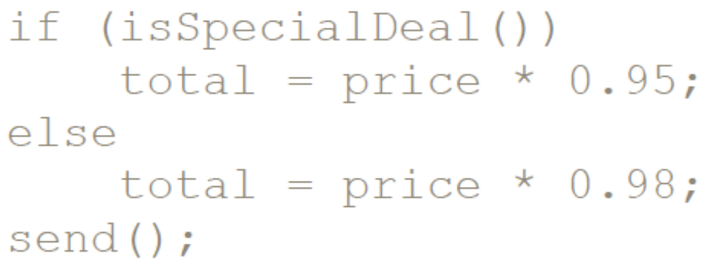
Consolidate Duplicate Conditional Fragments

- **Issue:** The same fragment of code is in all branches of a conditional expression.
- **Refactoring:** Move it outside of the expression.
- **Example**



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

⇛

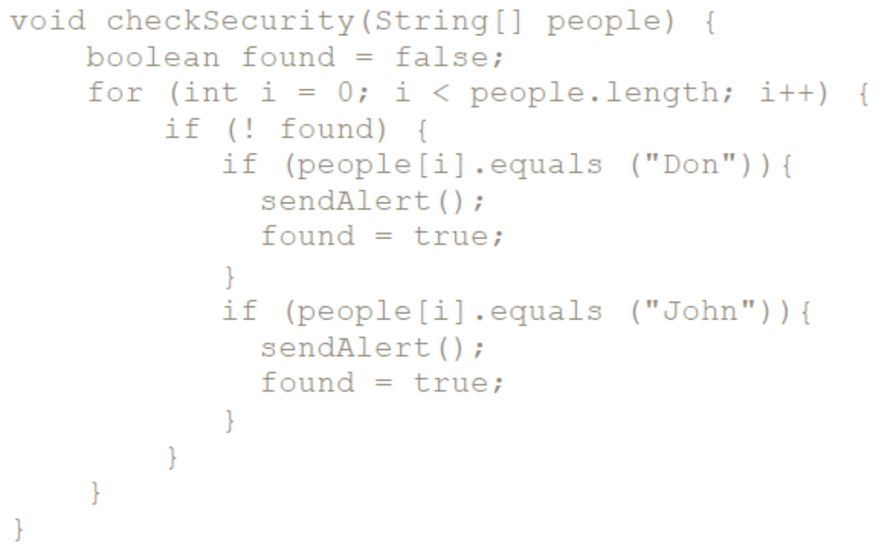


```
if (isSpecialDeal())
    total = price * 0.95;
else
    total = price * 0.98;
send();
```

Fowler, Martin. Refactoring: Improving the Design of Existing Code. Boston, MA, USA: Addison-Wesley, 1999.

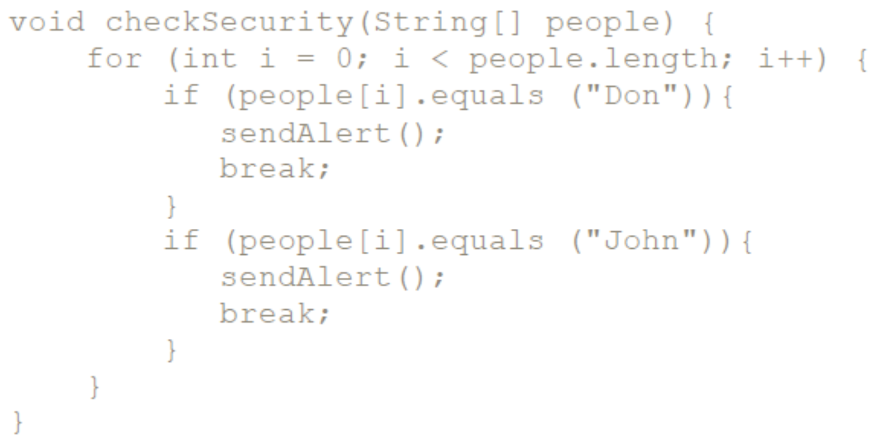
Remove Control Flag

- **Issue:** You have a variable that is acting as a control flag for a series of boolean expressions.
- **Refactoring:** Use a break or return instead.
- **Example**



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

⇒

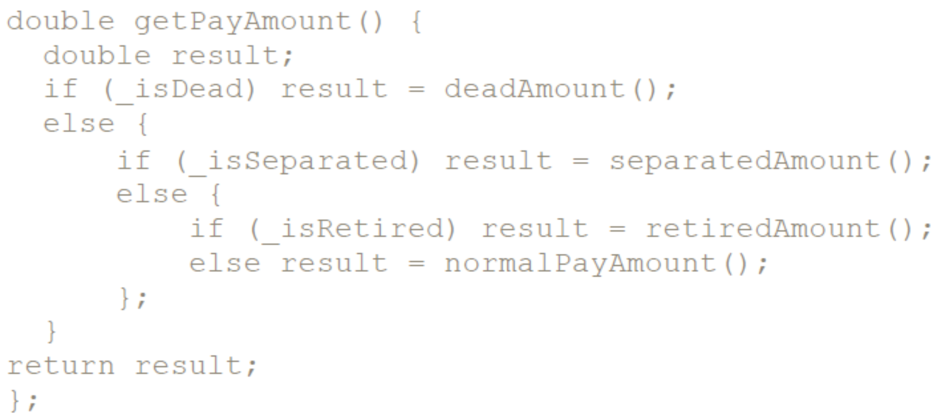


```
void checkSecurity(String[] people) {
    for (int i = 0; i < people.length; i++) {
        if (people[i].equals ("Don")){
           sendAlert();
           break;
        }
        if (people[i].equals ("John")){
           sendAlert();
           break;
        }
    }
}
```

Fowler, Martin. Refactoring: Improving the Design of Existing Code. Boston, MA, USA: Addison-Wesley, 1999.

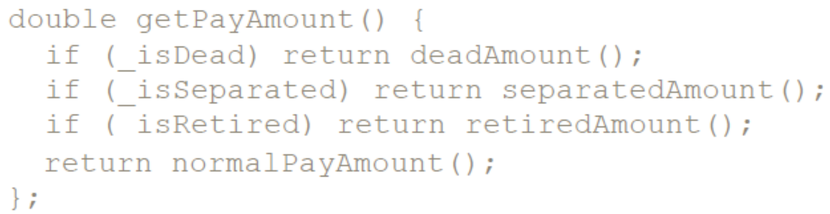
Replace Nested Conditional with Guard Clauses

- **Issue**: A method has conditional behavior that does not make clear the normal path of execution.
- **Refactoring**: Use guard clauses for all the special cases.
- **Example**



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

⇛

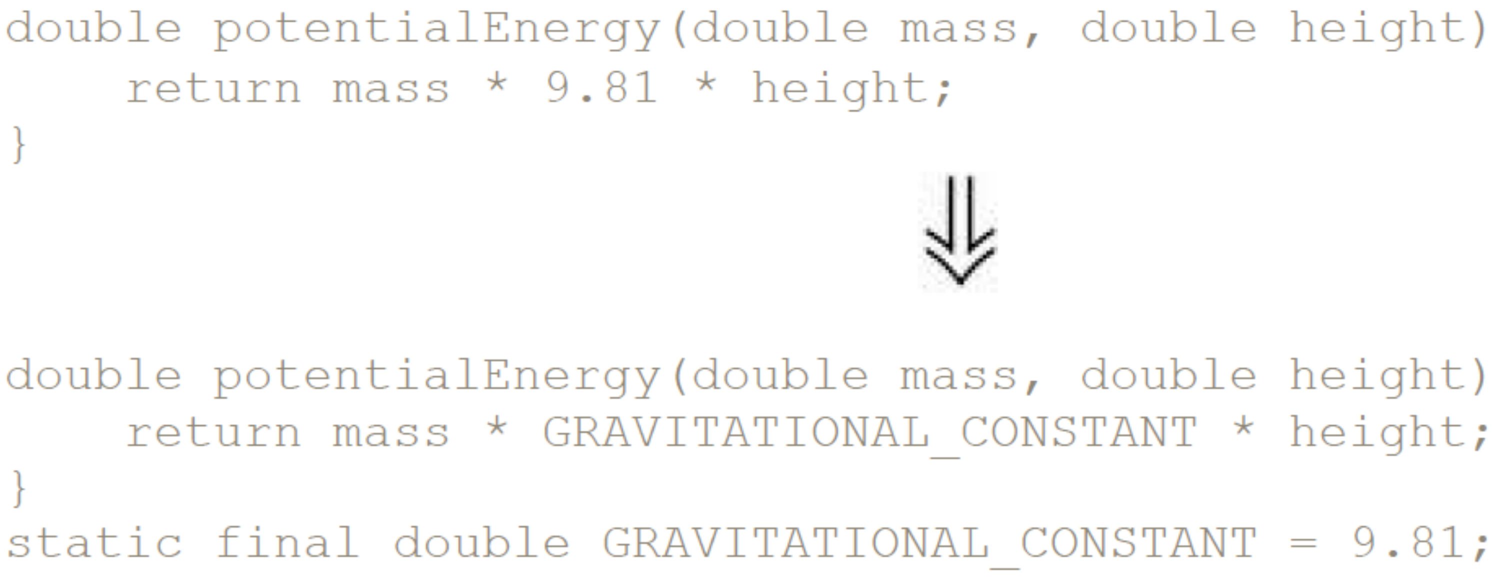


```
double getPayAmount() {
  if (_isDead) return deadAmount();
  if (_isSeparated) return separatedAmount();
  if ( isRetired) return retiredAmount();
  return normalPayAmount();
};
```

Fowler, Martin. Refactoring: Improving the Design of Existing Code. Boston, MA, USA: Addison-Wesley, 1999.

Replace Magic Number with Symbolic Constant

- **Issue**: You have a literal number with a particular meaning.
- **Refactoring**: Create a constant, name it after the meaning, and replace the number with it.
- **Example**



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

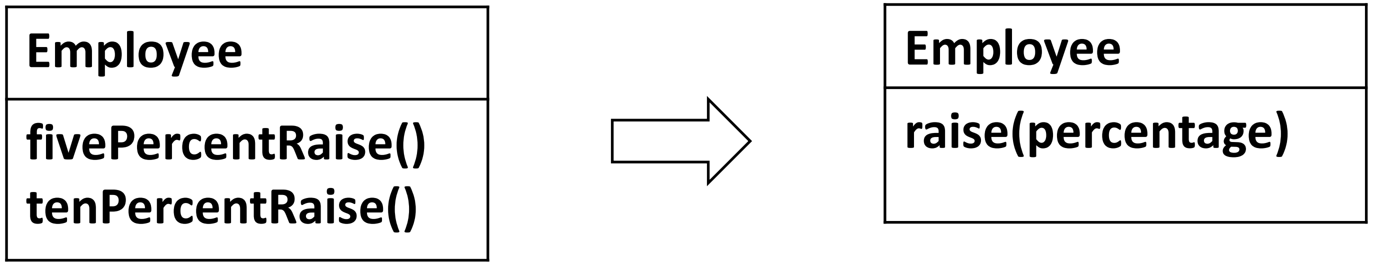
⇓

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

Fowler, Martin. Refactoring: Improving the Design of Existing Code. Boston, MA, USA: Addison-Wesley, 1999.

Parameterize Method

- **Issue:** Several methods do similar things but with different values contained in the method body.
- **Refactoring:** Create one method that uses a parameter for the different values. If the old methods are still used, their bodies should be replaced with a call to the new one.
- **Example**



Employee

fivePercentRaise()
tenPercentRaise()

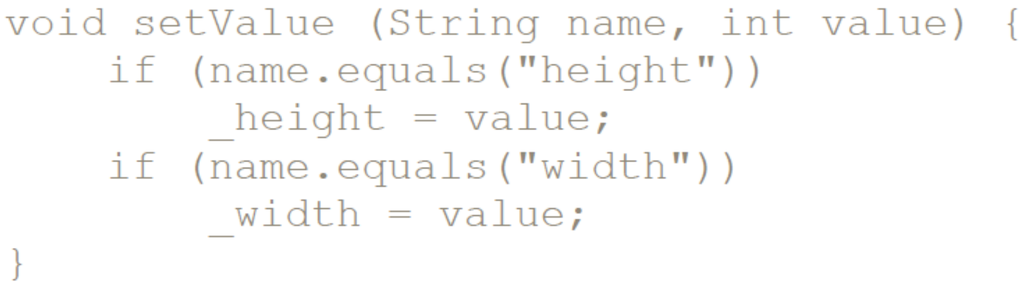
Employee

raise(percentage)

Fowler, Martin. Refactoring: Improving the Design of Existing Code. Boston, MA, USA: Addison-Wesley, 1999.

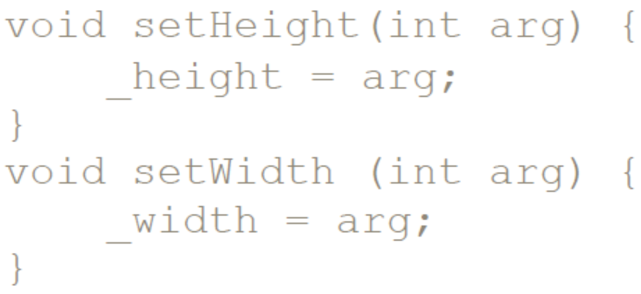
Replace Parameter with Explicit Methods

- **Issue**: You have a method that runs different code depending on the values of an enumerated parameter.
- **Refactoring**: Create a separate method for each value of the parameter.
- **Example**



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

⇒



```
void setHeight(int arg) {
    _height = arg;
}
void setWidth (int arg) {
    _width = arg;
}
```

Fowler, Martin. Refactoring: Improving the Design of Existing Code. Boston, MA, USA: Addison-Wesley, 1999.

Introduce Explaining Variable

- **Issue**: You have a complicated expression.
- **Refactoring**: Put the result of the expression, or parts of the expression, in a temporary variable with a name that explains the purpose.
- **Example**

```
double price() {
return _quantity * _itemPrice - Math.max(0, _quantity - 500) * _itemPrice * 0.05 + Math.min(_quantity * _itemPrice * 0.1, 100.0);
}
```

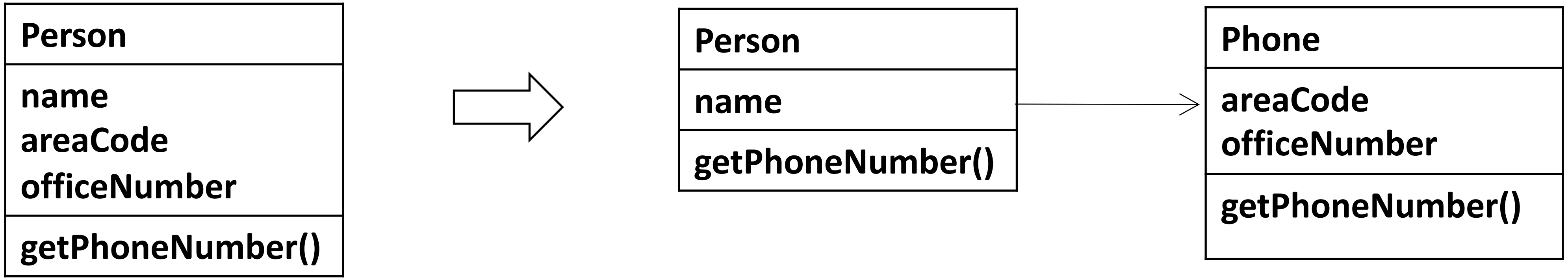
⇩

```
double price() {
final double basePrice = _quantity * _itemPrice;
final double quantityDiscount = Math.max(0, _quantity - 500) *_itemPrice * 0.05;
final double shipping = Math.min(basePrice * 0.1, 100.0);
return basePrice - quantityDiscount + shipping;
}
```

Fowler, Martin. Refactoring: Improving the Design of Existing Code. Boston, MA, USA: Addison-Wesley, 1999.

Extract Class

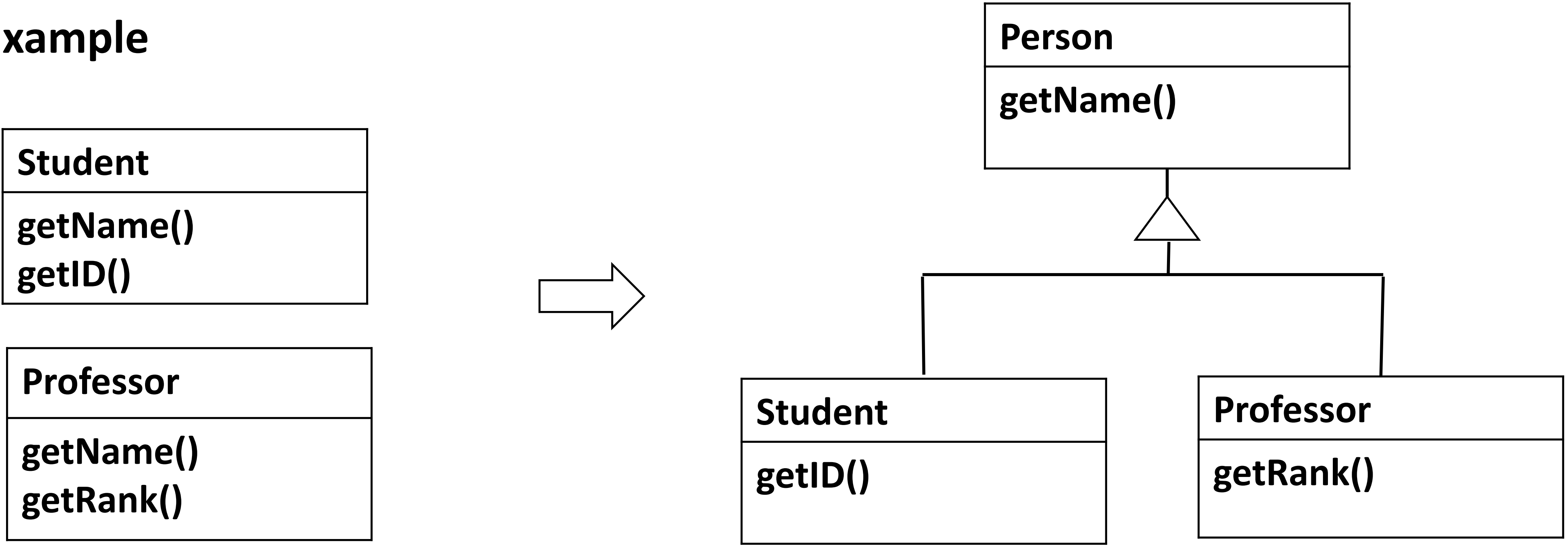
- **Issue**: You have one class doing work that should be done by two.
- **Refactoring**: Create a new class and move the relevant fields and methods from the old class into the new class.
- **Example**



Fowler, Martin. Refactoring: Improving the Design of Existing Code. Boston, MA, USA: Addison-Wesley, 1999.

Extract Superclass

- **Issue**: You have two classes with similar features.
- **Refactoring**: Create a superclass and move the common features to the superclass.
- **Example**



Student

getName()
getID()

Professor

getName()
getRank()

Person

getName()

Student

getID()

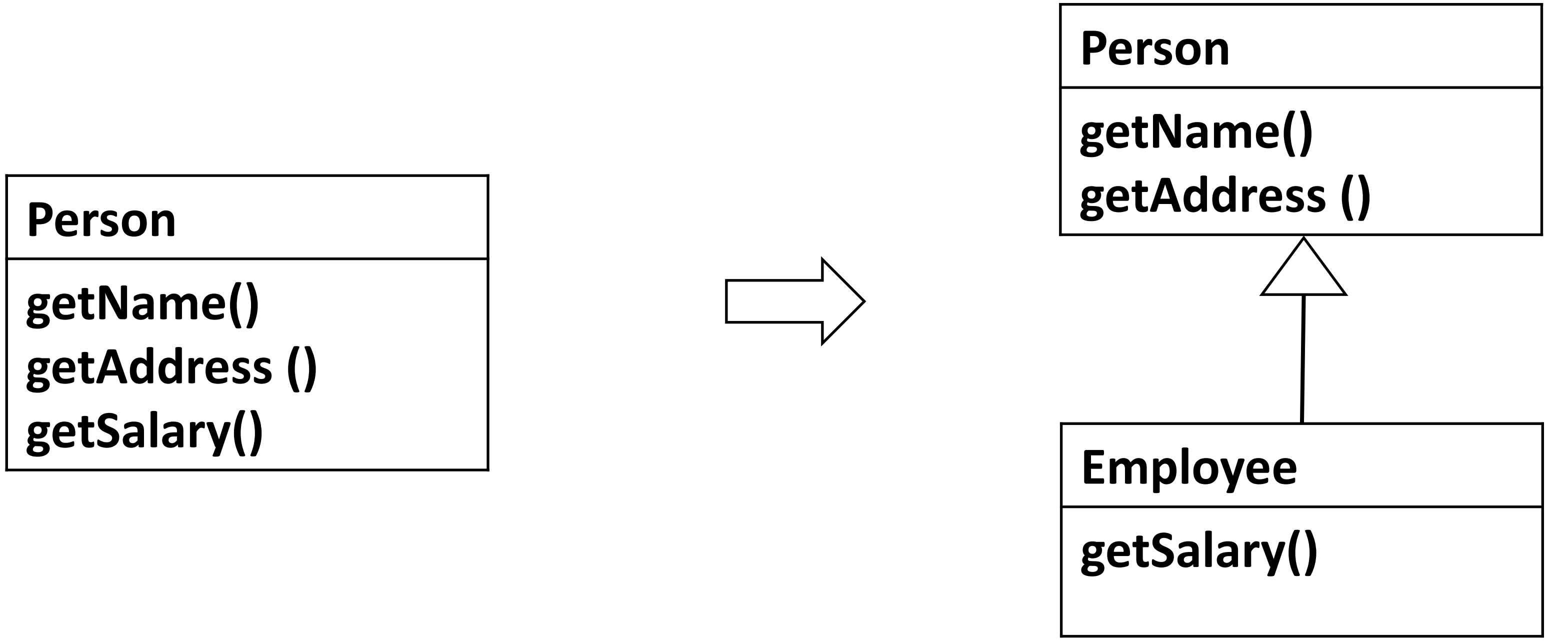
Professor

getRank()

Fowler, Martin. Refactoring: Improving the Design of Existing Code. Boston, MA, USA: Addison-Wesley, 1999.

Extract Subclass

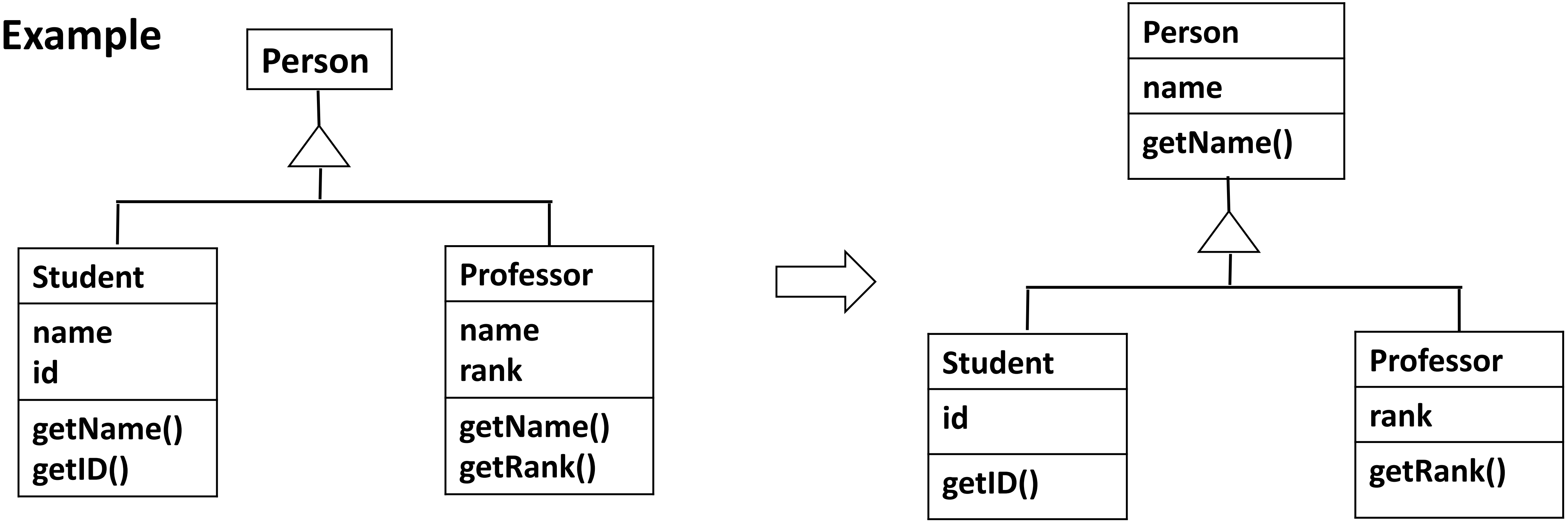
- **Issue**: A class has features that are used only in some instances.
- **Refactoring**: Create a subclass for that subset of features.
- **Example**



Fowler, Martin. Refactoring: Improving the Design of Existing Code. Boston, MA, USA: Addison-Wesley, 1999.

Pull up Field/Method

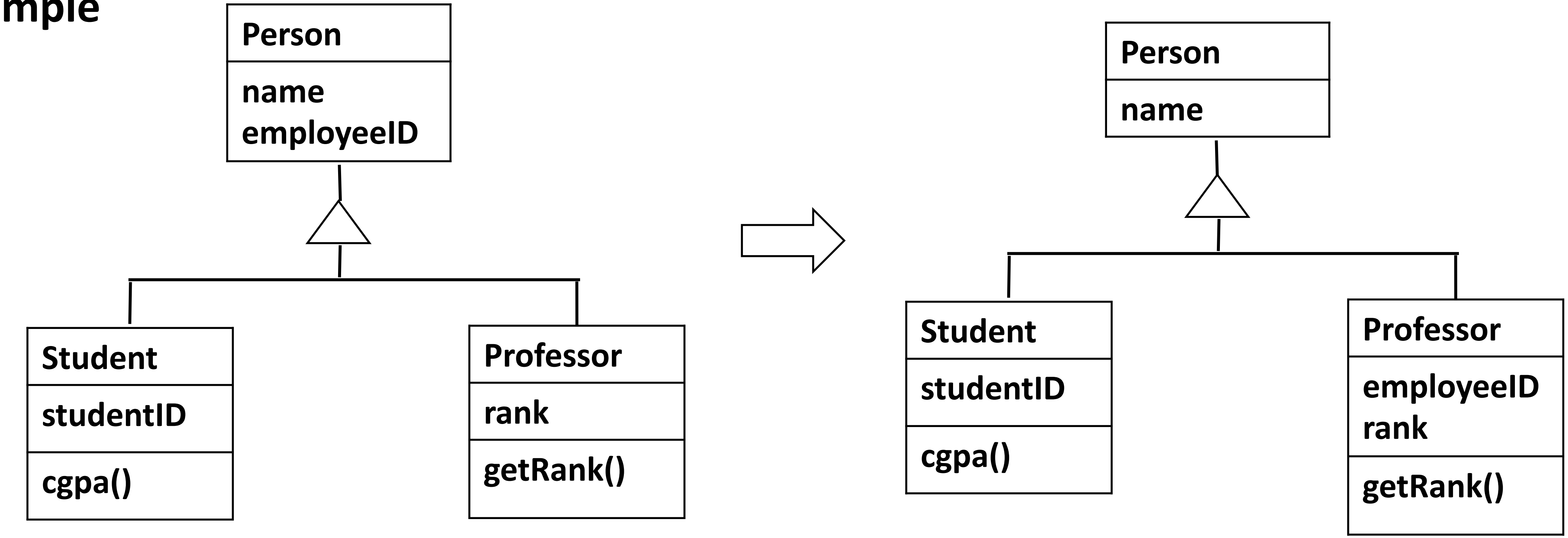
- **Issue**: You have methods with identical fields/methods.
- **Refactoring**: Move them to the superclass.
- **Example**



Fowler, Martin. Refactoring: Improving the Design of Existing Code. Boston, MA, USA: Addison-Wesley, 1999.

Push down Field/Method

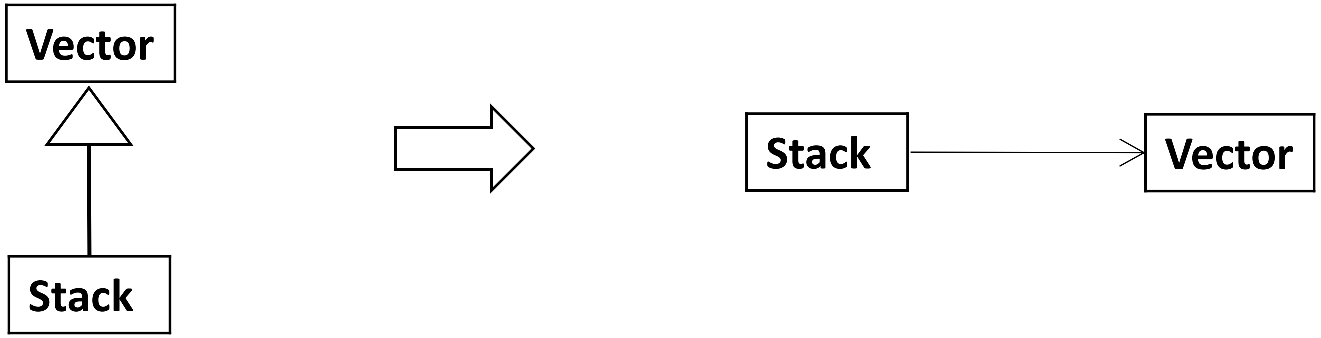
- **Issue:** Field/method of a superclass is relevant only for some of its subclasses.
- **Refactoring**: Move it to those subclasses.
- **Example**



Fowler, Martin. Refactoring: Improving the Design of Existing Code. Boston, MA, USA: Addison-Wesley, 1999.

Replace Inheritance with Delegation

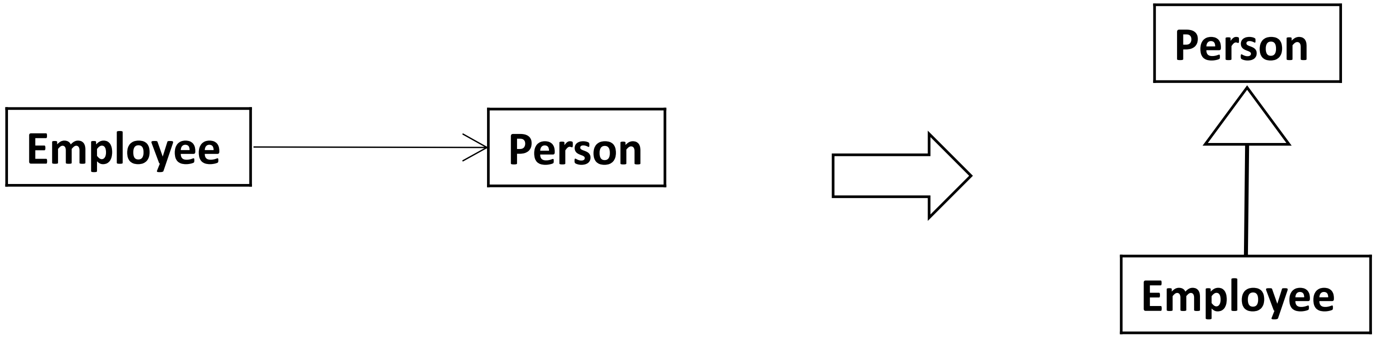
- **Issue**: A subclass uses only part of a superclasses interface or does not want to inherit data.
- **Refactoring**: Create a field for the superclass, adjust methods to delegate to the superclass, and remove the subclassing.
- **Example**



Fowler, Martin. Refactoring: Improving the Design of Existing Code. Boston, MA, USA: Addison-Wesley, 1999.

Replace Delegation with Inheritance

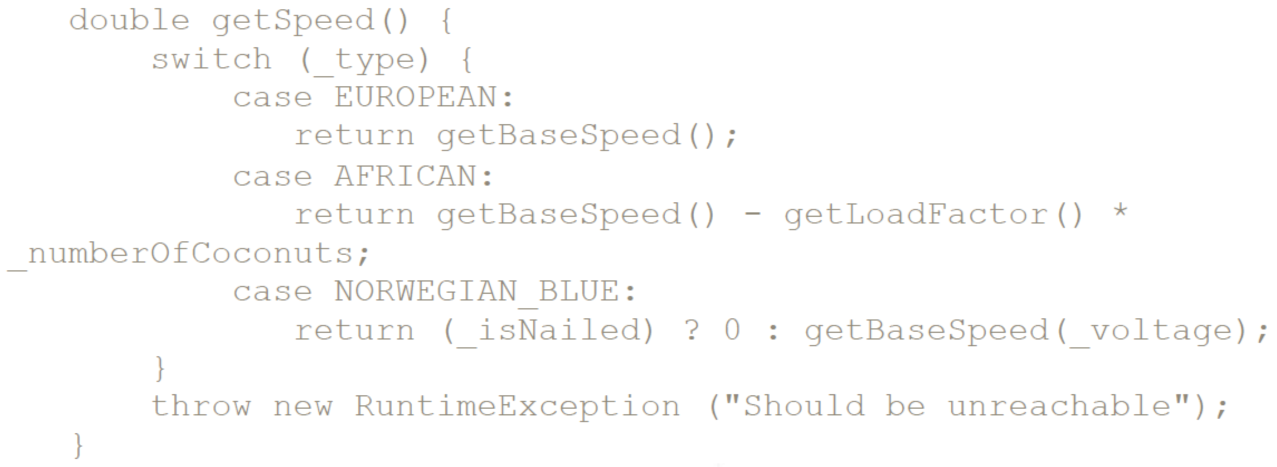
- **Issue**: You're using delegation and are often writing many simple delegations for the entire interface.
- **Refactoring**: Make the delegating class a subclass of the delegate.
- **Example**



Fowler, Martin. Refactoring: Improving the Design of Existing Code. Boston, MA, USA: Addison-Wesley, 1999.

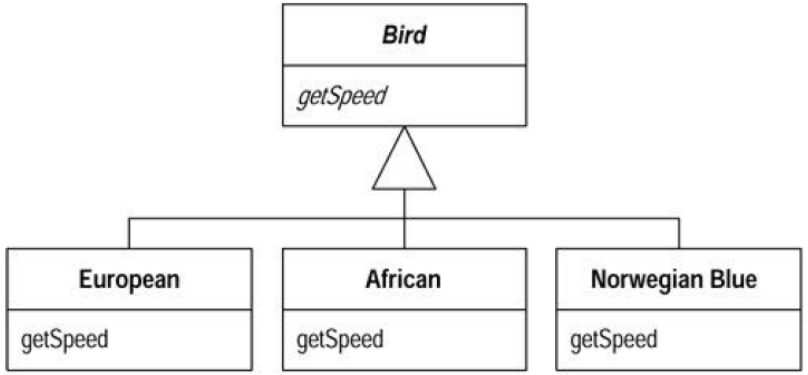
Replace Conditional with Polymorphism

- **Issue**: You have a conditional that chooses different behavior depending on the type of an object.
- **Refactoring**: Move each leg of the conditional to an overriding method in a subclass. Make the original method abstract.
- **Example**



```
   double getSpeed() {
       switch (_type) {
           case EUROPEAN:
              return getBaseSpeed();
           case AFRICAN:
              return getBaseSpeed() - getLoadFactor() *
_numberOfCoconuts;
           case NORWEGIAN_BLUE:
              return (_isNailed) ? 0 : getBaseSpeed(_voltage);
       }
       throw new RuntimeException ("Should be unreachable");
   }
```

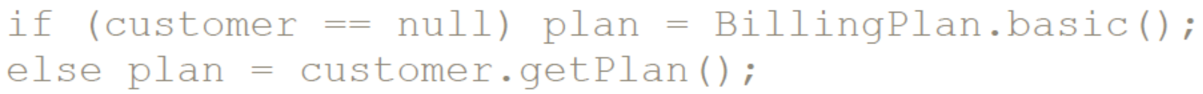
⇛



Fowler, Martin. Refactoring: Improving the Design of Existing Code. Boston, MA, USA: Addison-Wesley, 1999.

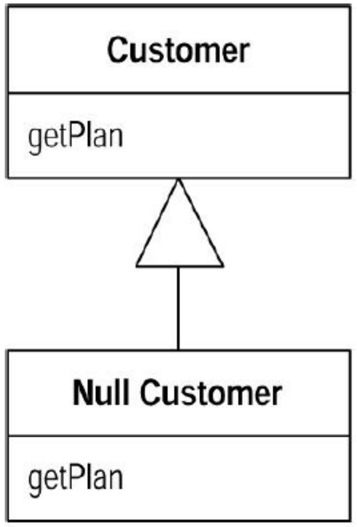
Introduce Null object

- **Issue**: You have repeated checks for a null value.
- **Refactoring**: Replace the null value with a null object.
- **Example**



```
if (customer == null) plan = BillingPlan.basic();
else plan = customer.getPlan();
```

⇛



Fowler, Martin. Refactoring: Improving the Design of Existing Code. Boston, MA, USA: Addison-Wesley, 1999.