Refactoring: Improving the Design of Existing Code

Chapter 6 - A First Set of Refactorings

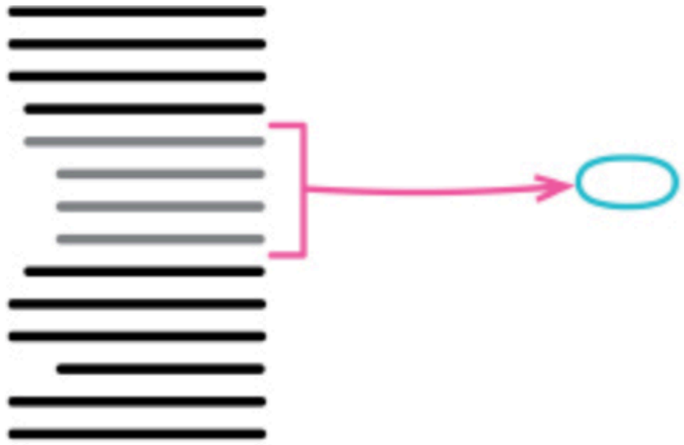
Lily Lin
2019.10.30

Outline

- EXTRACT FUNCTION
- INLINE FUNCTION
- EXTRACT VARIABLE
- INLINE VARIABLE
- CHANGE FUNCTION DECLARATION
- ENCAPSULATE VARIABLE
- RENAME VARIABLE
- INTRODUCE PARAMETER OBJECT
- COMBINE FUNCTIONS INTO CLASS
- COMBINE FUNCTIONS INTO TRANSFORM
- SPLIT PHASE

EXTRACT FUNCTION

- **Motivation**
  - The more lines found in a method, the harder it's to figure out what the method does.
  - Guideline: based on length, reuse or the separation between intention and implementation.
  - Concern: performance
  - Notes: only works if the name are good



https://refactoring.guru/extract-method

EXTRACT FUNCTION

- **Mechanics**
    1. Create a new function
    2. Copy and Paste the extracted code
    3. Pass the referenced variables as parameters
        - No Local Variables
        - Using Local Variables
        - Reassigning a Local Variable
    4. Replace the extracted code
- **IntelliJ**
    - ⌥⌘M

```
function printOwing(invoice) {
  printBanner();
  let outstanding  = calculateOutstanding();

  //print details
  console.log(`name: ${invoice.customer}`);
  console.log(`amount: ${outstanding}`);
}
```

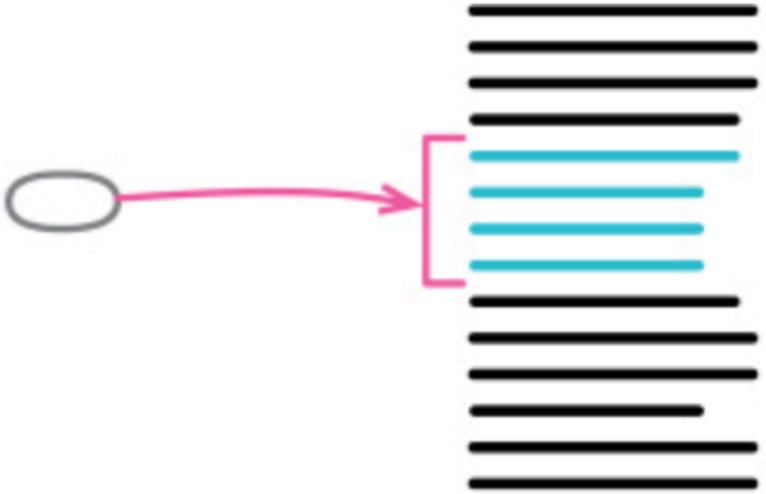
```
function printOwing(invoice) {
  printBanner();
  let outstanding  = calculateOutstanding();
  printDetails(outstanding);

  function printDetails(outstanding) {
    console.log(`name: ${invoice.customer}`);
    console.log(`amount: ${outstanding}`);
  }
}
```

https://www.jetbrains.com/help/idea/extract-method.html

INLINE FUNCTION

- **Motivation**
  - The body is as clear as the name.
  - Guideline:
    - using too much indirection
    - A group of functions that seem badly factor
  - Notes:
    - If you encounter these complexities, you shouldn’t do this refactoring.



https://refactoring.guru/inline-method

INLINE FUNCTION

- **Mechanics**

  1. Check this isn't a polymorphic method

  2. Find all the caller and replace them with function's body

  3. Remove the function

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

function moreThanFiveLateDeliveries(driver) {
  return driver.numberOfLateDeliveries > 5;
}
```

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

- **IntelliJ**

  - ⌥⌘N

https://www.jetbrains.com/help/idea/inline.html

EXTRACT VARIABLE

- **Motivation**
  - Expressions can become very complex and hard to read.
  - Guideline:
    - Condition of the if( ) operator or a part of the ?: operator in C-based languages
    - A long arithmetic expression without intermediate results
    - Long multipart lines
  - Concern: performance

https://refactoring.guru/extract-variable

EXTRACT VARIABLE

- **Mechanics**

  1. Check the expression doesn't have side effect
  2. Declare an immutable variable
  3. Replace the original expression with the new variable

```
return order.quantity * order.itemPrice -
  Math.max(0, order.quantity - 500) * order.itemPrice * 0.05 +
  Math.min(order.quantity * order.itemPrice * 0.1, 100);
```

⇓

```
const basePrice = order.quantity * order.itemPrice;
const quantityDiscount = Math.max(0, order.quantity - 500) * order.itemPrice * 0.05;
const shipping = Math.min(basePrice * 0.1, 100);
return basePrice - quantityDiscount + shipping;
```

- **IntelliJ**

  - ⌥⌘V

https://www.jetbrains.com/help/idea/extract-variable.html

INLINE VARIABLE

- **Motivation**
  - The name doesn’t really communicate more than the expression itself.
  - Concern: Sometimes seemingly useless temps are used to cache the result of an expensive operation that’s reused several times.

https://refactoring.guru/inline-temp

INLINE VARIABLE

- **Mechanics**

  1. Check right-hand side is free of side effect

  2. Repeat replacing references to the variable

  3. Remove variable

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

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

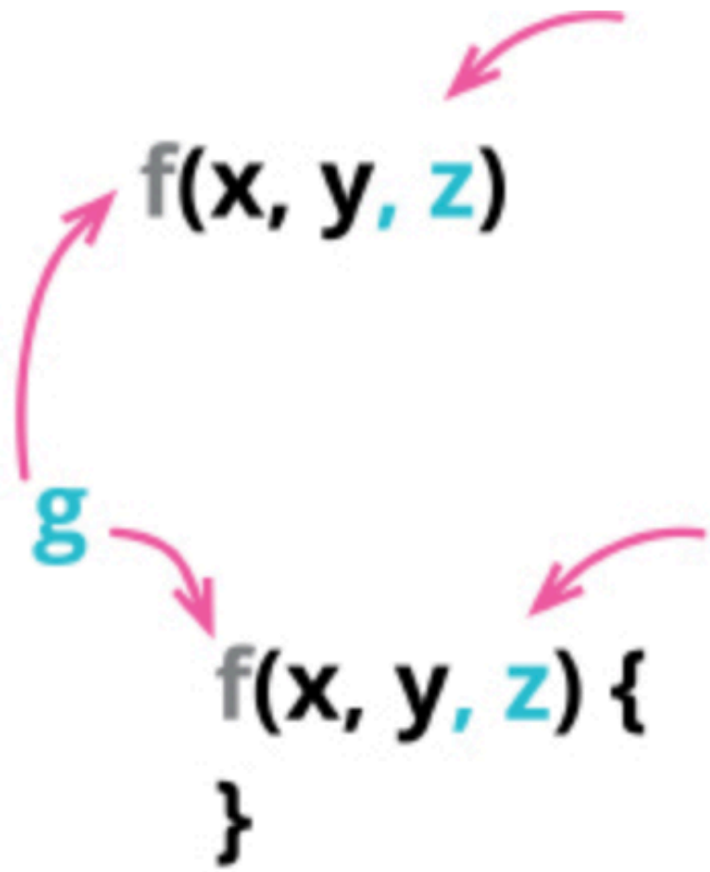
- IntelliJ

  - ⌥⌘N

https://www.jetbrains.com/help/idea/inline.html

CHANGE FUNCTION DECLARATION

- **Motivation**
  - Good joints
  - Guideline:
    - Rename
    - Add parameter
    - Remove parameter
  - Notes: There is no right answer, especially over time.



https://refactoring.guru/rename-method

https://refactoring.guru/add-parameter

https://refactoring.guru/remove-parameter

CHANGE FUNCTION DECLARATION

- **Simple Mechanics**
    1. Check it isn't referenced in the body of the function (for removing parameter)
    2. Change declaration
    3. Find all references and update
- **Migration Mechanics**
    1. Extract the body to the new function
    2. Apply inline function to old function
- **IntelliJ**
    - ⌘F6

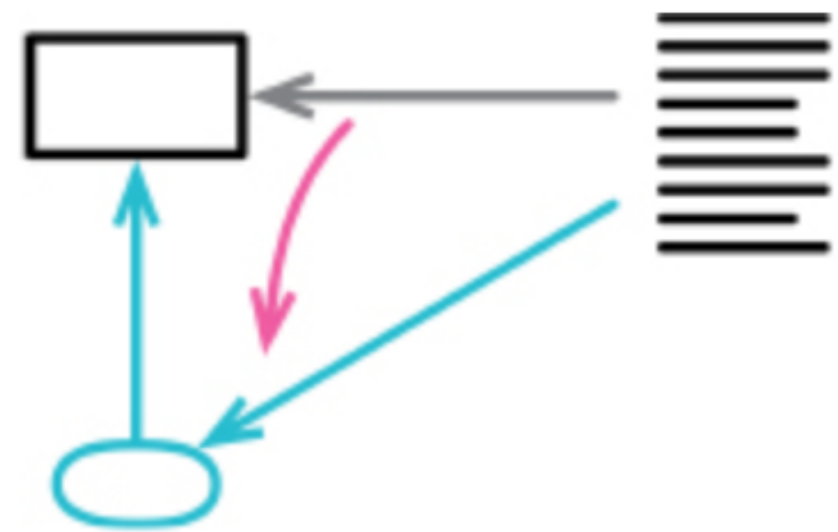
```
function circum(radius) {...}
```

```
function circumference(radius) {...}
```

https://www.jetbrains.com/help/idea/change-signature.html

ENCAPSULATE VARIABLE

- **Motivation**
  - It provides a clear point to monitor changes and use of the data
  - Guideline:
    - Make all mutable data encapsulated
  - Notes:
    - self-encapsulation: even internal references to fields within a class should go through accessor functions



https://refactoring.guru/encapsulate-field

ENCAPSULATE VARIABLE

- **Simple Mechanics**
  1. Check it isn't referenced in the body of the function (for removing parameter)
  2. Change declaration
  3. Find all references and update
- **Migration Mechanics**
  1. Extract the body to the new function
  2. Apply inline function to old function
- **IntelliJ**
  - Refactor | Encapsulate Fields

```
let defaultOwner = {firstName: "Martin", lastName: "Fowler"};
```

```
let defaultOwnerData = {firstName: "Martin", lastName: "Fowler"};
export function defaultOwner()       {return defaultOwnerData;}
export function setDefaultOwner(arg) {defaultOwnerData = arg;}
```

https://www.jetbrains.com/help/idea/encapsulate-fields.html

RENAME VARIABLE

- **Motivation**
  - Explain what I'm up to
  - Guideline:
    - The importance of a name depends on how widely it's used
  - Notes:
    - dynamically typed language like JavaScript, I do like to put the type into the name

RENAME VARIABLE

- **Mechanics**

  1. Find all references and change the name

     - If the variable is a published variable, you cannot do this refactoring.

```
let a = height * width;
```

```
let area = height * width;
```

- **IntelliJ**

  - ⇧F6

https://www.jetbrains.com/help/idea/rename-refactorings.html

INTRODUCE PARAMETER OBJECT

- **Motivation**
  - Reduces the size of parameter
  - Concern: Data Class (bad smell)
  - Notes: This process can change the conceptual picture of the code, raising these structures as new abstractions

https://refactoring.guru/introduce-parameter-object

INTRODUCE PARAMETER OBJECT

- **Mechanics**

  1. Create a new class
  2. Add a parameter for the new structure
  3. Adjust caller
  4. Replace the original parameter

```
function amountInvoiced(startDate, endDate) {...}
function amountReceived(startDate, endDate) {...}
function amountOverdue(startDate, endDate) {...}
```

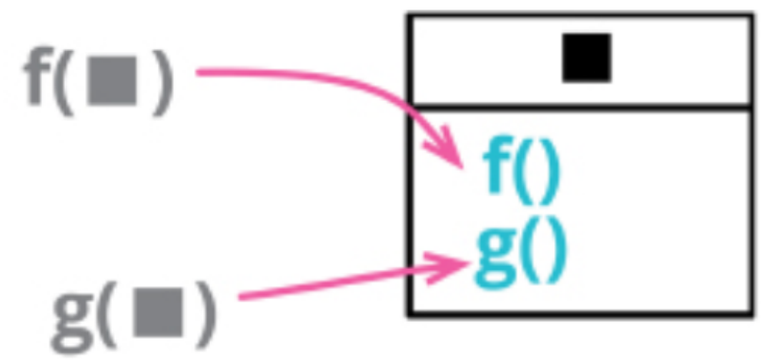
```
function amountInvoiced(aDateRange) {...}
function amountReceived(aDateRange) {...}
function amountOverdue(aDateRange) {...}
```

- **IntelliJ**

  - Refactor| Parameter Object

COMBINE FUNCTIONS INTO CLASS

- **Motivation**
  - Functions that operate closely together on a common body of data



COMBINE FUNCTIONS INTO CLASS

- **Mechanics**

  1. Apply *Encapsulate Record* to the common data record (use *Introduce Parameter Object* to create a record to group it together)

  2. Move Function into the new class F6

  3. Any arguments to the function call that are members can be removed

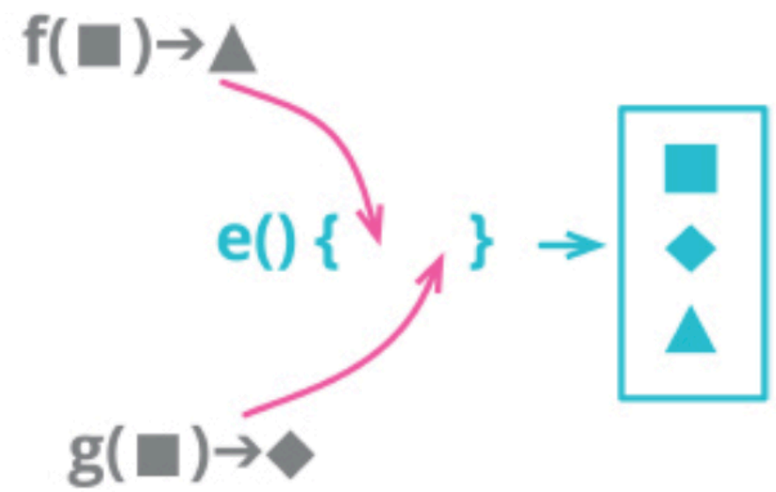
  4. Any logic that manipulates the data can be extracted and moved into the new class.

```
function base(aReading) {...}
function taxableCharge(aReading) {...}
function calculateBaseCharge(aReading) {...}
```

```
class Reading {
  base() {...}
  taxableCharge() {...}
  calculateBaseCharge() {...}
}
```

COMBINE FUNCTIONS INTO TRANSFORM

- **Motivation**
  - Bring all of these derivations together
  - Notes:
    - vs. Combine functions into class
      - Takes the source data as input and calculates all the derivations
      - Depends more on the broader context of the program
      - Inconsistencies when source data updating



COMBINE FUNCTIONS INTO TRANSFORM

- **Mechanics**

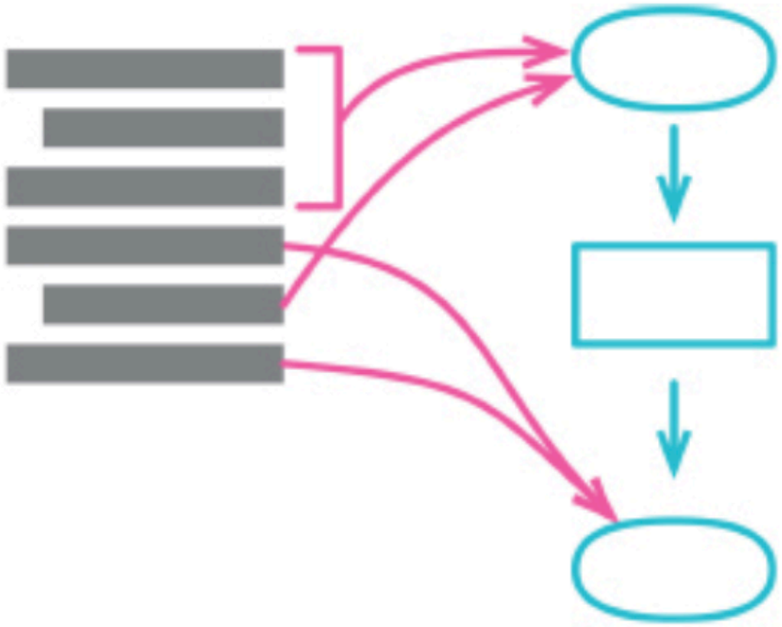
  1. Create a transformation function (take the record and return the same value)
  2. Move the logic into function body
  3. Change the client code

```
function base(aReading) {...}
function taxableCharge(aReading) {...}
```

```
function enrichReading(argReading) {
  const aReading = _.cloneDeep(argReading);
  aReading.baseCharge = base(aReading);
  aReading.taxableCharge = taxableCharge(aReading);
  return aReading;
}
```

SPLIT PHASE

- **Motivation**
  - Split code into separate modules when dealing with different things
  - Guideline:
    - Different stages of the fragment use different sets of data and functionsNotes:



https://refactoring.guru/split-temporary-variable

SPLIT PHASE

- **Mechanics**
  1. Extract the second phase code into its own function
  2. Introduce an intermediate data
  3. Extract the first phase

```
const orderData = orderString.split(/\s+/);
const productPrice = priceList[orderData[0].split("-")[1]];
const orderPrice = parseInt(orderData[1]) * productPrice;
```

```
const orderRecord = parseOrder(order);
const orderPrice = price(orderRecord, priceList);

function parseOrder(aString) {
  const values =  aString.split(/\s+/);
  return ({
    productID: values[0].split("-")[1],
    quantity: parseInt(values[1]),
  });
}
function price(order, priceList) {
  return order.quantity * priceList[order.productID];
}
```

References

- Refactoring GURU
  https://refactoring.guru/
  https://github.com/RefactoringGuru/refactoring-examples

- IntelliJ INDEA - Refactoring code
  https://www.jetbrains.com/help/idea/refactoring-source-code.html