

# 重构

## 改善既有代码的设计

(第2版)

REFACTORING

[美] 马丁·福勒 (Martin Fowler) 著  
熊节 林从羽 译

Improving the Design of Existing Code  
SECOND EDITION



## 00

00	1.1
00	1.2
01 00 00000000	1.3
1.1 00	1.3.1
1.2 0000000000	1.3.2
1.3 00000000	1.3.3
1.4 00 statement 00	1.3.4
00 play 00	1.3.4.1
00 format 00	1.3.4.2
0000000000	1.3.4.3
1.5 0000000000	1.3.5
1.6 000000000000	1.3.6
1.7 000000000000000000	1.3.7
1.8 0000000000	1.3.8
0000000000	1.3.8.1
00000000000000	1.3.8.2
1.9 0000000000000000	1.3.9
1.10 00	1.3.10
02 00 000000	1.4
2.1 00000	1.4.1
2.2 00000	1.4.2
2.3 00000	1.4.3
0000000000	1.4.3.1
000000000000	1.4.3.2
0000000 bug	1.4.3.3
0000000000	1.4.3.4
2.4 00000	1.4.4
000000000000000000	1.4.4.1
000000000000000000	1.4.4.2
00000000	1.4.4.3
000000000000000000	1.4.4.4
00000	1.4.4.5
0000000000	1.4.4.6
00000000	1.4.4.7
0000000000	1.4.4.8

2.5	□□□□□	1.4.5
	□□□□□□□	1.4.5.1
	□□□□□	1.4.5.2
	□□	1.4.5.3
	□□	1.4.5.4
	□□□□	1.4.5.5
	□□□	1.4.5.6
2.6	□□□□□□ YAGNI	1.4.6
2.7	□□□□□□□□□□	1.4.7
2.8	□□□□□□	1.4.8
2.9	□□□□□□□	1.4.9
2.10	□□□□□□	1.4.10
2.11	□□□□□	1.4.11
□ 3 □	□□□□□□□	1.5
3.1	□□□□□Mysterious Name□	1.5.1
3.2	□□□□□Duplicated Code□	1.5.2
3.3	□□□□□Long Function□	1.5.3
3.4	□□□□□□□Long Parameter List□	1.5.4
3.5	□□□□□Global Data□	1.5.5
3.6	□□□□□Mutable Data□	1.5.6
3.7	□□□□□Divergent Change□	1.5.7
3.8	□□□□□Shotgun Surgery□	1.5.8
3.9	□□□□□Feature Envy□	1.5.9
3.10	□□□□□Data Clumps□	1.5.10
3.11	□□□□□□□Primitive Obsession□	1.5.11
3.12	□□□ switch □Repeated Switches□	1.5.12
3.13	□□□□□Loops□	1.5.13
3.14	□□□□□□□Lazy Element□	1.5.14
3.15	□□□□□□□□Speculative Generality□	1.5.15
3.16	□□□□□Temporary Field□	1.5.16
3.17	□□□□□□□Message Chains□	1.5.17
3.18	□□□□□Middle Man□	1.5.18
3.19	□□□□□Insider Trading□	1.5.19
3.20	□□□□□Large Class□	1.5.20
3.21	□□□□□□□Alternative Classes with Different Interfaces□	1.5.21
3.22	□□□□□Data Class□	1.5.22
3.23	□□□□□□□Refused Bequest□	1.5.23
3.24	□□□□□Comments□	1.5.24

□ 4 □ □□□□□	1.6
4.1 □□□□□□	1.6.1
4.2 □□□□□□	1.6.2
4.3 □□□□	1.6.3
4.4 □□□□□□	1.6.4
4.5 □□□□□	1.6.5
4.6 □□□□□	1.6.6
4.7 □□□□□□	1.6.7
□ 5 □ □□□□□	1.7
5.1 □□□□□□	1.7.1
5.2 □□□□□□	1.7.2
□ 6 □ □□□□□	1.8
6.1 □□□□Extract Function□	1.8.1
□□	1.8.1.1
□□	1.8.1.2
□□□□□□□	1.8.1.3
□□□□□□□	1.8.1.4
□□□□□□□□□	1.8.1.5
6.2 □□□□Inline Function□	1.8.2
□□	1.8.2.1
□□	1.8.2.2
□□	1.8.2.3
6.3 □□□□Extract Variable□	1.8.3
□□	1.8.3.1
□□	1.8.3.2
□□	1.8.3.3
□□□□□□□	1.8.3.4
6.4 □□□□Inline Variable□	1.8.4
□□	1.8.4.1
□□	1.8.4.2
6.5 □□□□□□Change Function Declaration□	1.8.5
□□	1.8.5.1
□□	1.8.5.2
□□□□	1.8.5.3
□□□□	1.8.5.4
□□□□□□□□□□	1.8.5.5
□□□□□□□□□□	1.8.5.6
□□□□□□	1.8.5.7

□□□□□□□□	1.8.5.8
6.6 □□□□Encapsulate Variable□	1.8.6
□□	1.8.6.1
□□	1.8.6.2
□□	1.8.6.3
□□□	1.8.6.4
6.7 □□□□Rename Variable□	1.8.7
□□	1.8.7.1
□□	1.8.7.2
□□	1.8.7.3
□□□□	1.8.7.4
6.8 □□□□□□Introduce Parameter Object□	1.8.8
□□	1.8.8.1
□□	1.8.8.2
□□	1.8.8.3
6.9 □□□□□□Combine Functions into Class□	1.8.9
□□	1.8.9.1
□□	1.8.9.2
□□	1.8.9.3
6.10 □□□□□□Combine Functions into Transform□	1.8.10
□□	1.8.10.1
□□	1.8.10.2
□□	1.8.10.3
6.11 □□□□Split Phase□	1.8.11
□□	1.8.11.1
□□	1.8.11.2
□□	1.8.11.3
□ 7 □ □□	1.9
7.1 □□□□Encapsulate Record□	1.9.1
□□	1.9.1.1
□□	1.9.1.2
□□	1.9.1.3
□□□□□□□□	1.9.1.4
7.2 □□□□Encapsulate Collection□	1.9.2
□□	1.9.2.1
□□	1.9.2.2
□□	1.9.2.3
7.3 □□□□□□□□Replace Primitive with Object□	1.9.3

□□	1.9.3.1
□□	1.9.3.2
□□	1.9.3.3
7.4 □□□□□□□□Replace Temp with Query□	1.9.4
□□	1.9.4.1
□□	1.9.4.2
□□	1.9.4.3
7.5 □□□□Extract Class□	1.9.5
□□	1.9.5.1
□□	1.9.5.2
□□	1.9.5.3
7.6 □□□□Inline Class□	1.9.6
□□	1.9.6.1
□□	1.9.6.2
□□	1.9.6.3
7.7 □□□□□□Hide Delegate□	1.9.7
□□	1.9.7.1
□□	1.9.7.2
□□	1.9.7.3
7.8 □□□□□□Remove Middle Man□	1.9.8
□□	1.9.8.1
□□	1.9.8.2
□□	1.9.8.3
7.9 □□□□□Substitute Algorithm□	1.9.9
□□	1.9.9.1
□□	1.9.9.2
□ 8 □ □□□□	1.10
8.1 □□□□□Move Function□	1.10.1
□□	1.10.1.1
□□	1.10.1.2
□□□□□□□□□□	1.10.1.3
□□□□□□□□□□	1.10.1.4
8.2 □□□□□Move Field□	1.10.2
□□	1.10.2.1
□□	1.10.2.2
□□	1.10.2.3
□□□□□□□□□□	1.10.2.4
8.3 □□□□□□□Move Statements into Function□	1.10.3

□□	1.10.3.1
□□	1.10.3.2
□□	1.10.3.3
8.4 □□□□□□□□□□Move Statements to Callers□	1.10.4
□□	1.10.4.1
□□	1.10.4.2
□□	1.10.4.3
8.5 □□□□□□□□□□Replace Inline Code with Function Call□	1.10.5
□□	1.10.5.1
□□	1.10.5.2
8.6 □□□□□Slide Statements□	1.10.6
□□	1.10.6.1
□□	1.10.6.2
□□	1.10.6.3
□□□□□□□□□□	1.10.6.4
8.7 □□□□□Split Loop□	1.10.7
□□	1.10.7.1
□□	1.10.7.2
□□	1.10.7.3
8.8 □□□□□□□□Replace Loop with Pipeline□	1.10.8
□□	1.10.8.1
□□	1.10.8.2
□□	1.10.8.3
8.9 □□□□□Remove Dead Code□	1.10.9
□□	1.10.9.1
□□	1.10.9.2
□ 9 □ □□□□□	1.11
9.1 □□□□□Split Variable□	1.11.1
□□	1.11.1.1
□□	1.11.1.2
□□	1.11.1.3
□□□□□□□□□□	1.11.1.4
9.2 □□□□□Rename Field□	1.11.2
□□	1.11.2.1
□□	1.11.2.2
□□□□□□□□	1.11.2.3
9.3 □□□□□□□□□□Replace Derived Variable with Query□	1.11.3
□□	1.11.3.1

□□	1.11.3.2
□□	1.11.3.3
□□□□□□□□	1.11.3.4
9.4 □□□□□□□□Change Reference to Value□	1.11.4
□□	1.11.4.1
□□	1.11.4.2
□□	1.11.4.3
9.5 □□□□□□□□Change Value to Reference□	1.11.5
□□	1.11.5.1
□□	1.11.5.2
□□	1.11.5.3
□ 10 □ □□□□□	1.12
10.1 □□□□□□Decompose Conditional□	1.12.1
□□	1.12.1.1
□□	1.12.1.2
□□	1.12.1.3
10.2 □□□□□□Consolidate Conditional Expression□	1.12.2
□□	1.12.2.1
□□	1.12.2.2
□□	1.12.2.3
□□□□□□	1.12.2.4
10.3 □□□□□□□□Replace Nested Conditional with Guard Clauses□	1.12.3
□□	1.12.3.1
□□	1.12.3.2
□□	1.12.3.3
□□□□□□	1.12.3.4
10.4 □□□□□□□□Replace Conditional with Polymorphism□	1.12.4
□□	1.12.4.1
□□	1.12.4.2
□□	1.12.4.3
□□□□□□□□	1.12.4.4
10.5 □□□□Introduce Special Case□	1.12.5
□□	1.12.5.1
□□	1.12.5.2
□□	1.12.5.3
□□□□□□□□	1.12.5.4
□□□□□□	1.12.5.5
10.6 □□□□Introduce Assertion□	1.12.6



□□	1.12.6.1
□□	1.12.6.2
□□	1.12.6.3
□ 11 □ □□ API	1.13
11.1 □□□□□□□□□□ Separate Query from Modifier□	1.13.1
□□	1.13.1.1
□□	1.13.1.2
□□	1.13.1.3
11.2 □□□□□□ Parameterize Function□	1.13.2
□□	1.13.2.1
□□	1.13.2.2
□□	1.13.2.3
11.3 □□□□□□ Remove Flag Argument□	1.13.3
□□	1.13.3.1
□□	1.13.3.2
□□	1.13.3.3
11.4 □□□□□□ Preserve Whole Object□	1.13.4
□□	1.13.4.1
□□	1.13.4.2
□□	1.13.4.3
□□□□□□□□□□	1.13.4.4
11.5 □□□□□□ Replace Parameter with Query□	1.13.5
□□	1.13.5.1
□□	1.13.5.2
□□	1.13.5.3
11.6 □□□□□□ Replace Query with Parameter□	1.13.6
□□	1.13.6.1
□□	1.13.6.2
□□	1.13.6.3
11.7 □□□□□□ Remove Setting Method□	1.13.7
□□	1.13.7.1
□□	1.13.7.2
□□	1.13.7.3
11.8 □□□□□□□□□□ Replace Constructor with Factory Function□	1.13.8
□□	1.13.8.1
□□	1.13.8.2
□□	1.13.8.3
11.9 □□□□□□ Replace Function with Command□	1.13.9

□□	1.13.9.1
□□	1.13.9.2
□□	1.13.9.3
11.10 □□□□□□Replace Command with Function□	1.13.10
□□	1.13.10.1
□□	1.13.10.2
□□	1.13.10.3
□ 12 □ □□□□□	1.14
12.1 □□□□Pull Up Method□	1.14.1
□□	1.14.1.1
□□	1.14.1.2
□□	1.14.1.3
12.2 □□□□Pull Up Field□	1.14.2
□□	1.14.2.1
□□	1.14.2.2
12.3 □□□□□□Pull Up Constructor Body□	1.14.3
□□	1.14.3.1
□□	1.14.3.2
□□	1.14.3.3
12.4 □□□□Push Down Method□	1.14.4
□□	1.14.4.1
□□	1.14.4.2
12.5 □□□□Push Down Field□	1.14.5
□□	1.14.5.1
□□	1.14.5.2
12.6 □□□□□□Replace Type Code with Subclasses□	1.14.6
□□	1.14.6.1
□□	1.14.6.2
□□	1.14.6.3
□□□□□□	1.14.6.4
12.7 □□□□Remove Subclass□	1.14.7
□□	1.14.7.1
□□	1.14.7.2
□□	1.14.7.3
12.8 □□□□Extract Superclass□	1.14.8
□□	1.14.8.1
□□	1.14.8.2
□□	1.14.8.3

12.9 □□□□□□Collapse Hierarchy□	1.14.9
□□	1.14.9.1
□□	1.14.9.2
12.10 □□□□□□Replace Subclass with Delegate□	1.14.10
□□	1.14.10.1
□□	1.14.10.2
□□	1.14.10.3
□□□□□□□□	1.14.10.4
12.11 □□□□□□Replace Superclass with Delegate□	1.14.11
□□	1.14.11.1
□□	1.14.11.2
□□	1.14.11.3

# book-refactoring2



book-refactoring2

## book-refactoring2

- 地址: <https://book-refactoring2.ifmicro.com>
- 格式: pdf, epub, mobi

## book-refactoring2/README

### 安装

1. 安装 book-refactoring2
2. 安装 book-refactoring2

```
$ git clone https://github.com/MwumLi/book-refactoring2.git
$ npm i
```

### 安装

安装 book-refactoring2, 安装 book-refactoring2, 安装 book-refactoring2

安装 book-refactoring2:

```
$ npm run build
```

安装 book-refactoring2 \_book/ 安装, 安装 book-refactoring2

### 安装

安装 book-refactoring2, 安装 book-refactoring2 mobi , epub 安装 pdf 安装 book-refactoring2:

```
$ npm run ebook
```

安装: 安装 book-refactoring2 calibre, 安装 gitbook 安装 book-refactoring2

## book-refactoring2

- Node.js ^10.x - ^11.x LTS 安装
- gitbook ^3.x: 安装 book-refactoring2

## book-refactoring2







2000 年，Google 公司推出了 JavaScript 的虚拟机，即 V8 引擎。V8 引擎的推出，使得 JavaScript 的运行速度得到了极大的提升，从而使得 JavaScript 在 Web 开发中的地位日益重要。

在 2000 年之前，JavaScript 的运行速度非常慢，这主要是因为 JavaScript 的虚拟机（V8 引擎）还没有出现。

- 在 2000 年之前，JavaScript 的运行速度非常慢，这主要是因为 JavaScript 的虚拟机（V8 引擎）还没有出现。
- 在 2000 年之前，JavaScript 的运行速度非常慢，这主要是因为 JavaScript 的虚拟机（V8 引擎）还没有出现。
- 在 2000 年之前，JavaScript 的运行速度非常慢，这主要是因为 JavaScript 的虚拟机（V8 引擎）还没有出现。
- 在 2000 年之前，JavaScript 的运行速度非常慢，这主要是因为 JavaScript 的虚拟机（V8 引擎）还没有出现。

在 2000 年之前，JavaScript 的运行速度非常慢，这主要是因为 JavaScript 的虚拟机（V8 引擎）还没有出现。

## 2000 年

在 2000 年之前，JavaScript 的运行速度非常慢，这主要是因为 JavaScript 的虚拟机（V8 引擎）还没有出现。

在 2000 年之前，JavaScript 的运行速度非常慢，这主要是因为 JavaScript 的虚拟机（V8 引擎）还没有出现。

Ralph Johnson 的 UIUC 项目，以及 Kent Beck 的 Kent 项目，以及 Bill Opdyke 的 Bill Opdyke 项目，以及 John Brant 的 John Brant 项目，以及 Don Roberts 的 Don Roberts 项目，以及 Refactoring Browser 项目，以及 Smalltalk 项目。

在 2000 年之前，JavaScript 的运行速度非常慢，这主要是因为 JavaScript 的虚拟机（V8 引擎）还没有出现。

## 2000 年

在 2000 年之前，JavaScript 的运行速度非常慢，这主要是因为 JavaScript 的虚拟机（V8 引擎）还没有出现。

在 2000 年之前，JavaScript 的运行速度非常慢，这主要是因为 JavaScript 的虚拟机（V8 引擎）还没有出现。

在 2000 年之前，JavaScript 的运行速度非常慢，这主要是因为 JavaScript 的虚拟机（V8 引擎）还没有出现。







□□

```
[
  {
    "customer": "BigCo",
    "performances": [
      {
        "playID": "hamlet",
        "audience": 55
      },
      {
        "playID": "as-like",
        "audience": 35
      },
      {
        "playID": "othello",
        "audience": 40
      }
    ]
  }
]
```

□□□□□□□□□□□□□□□□□□

```
function statement (invoice, plays) {
  let totalAmount = 0;
  let volumeCredits = 0;
  let result = `Statement for ${invoice.customer}\n`;
  const format = new Intl.NumberFormat("en-US",
    { style: "currency", currency: "USD",
      minimumFractionDigits: 2 }).format;
  for (let perf of invoice.performances) {
    const play = plays[perf.playID];
    let thisAmount = 0;

    switch (play.type) {
      case "tragedy":
        thisAmount = 40000;
        if (perf.audience > 30) {
          thisAmount += 1000 * (perf.audience - 30);
        }
        break;
      case "comedy":
        thisAmount = 30000;
        if (perf.audience > 20) {
          thisAmount += 10000 + 500 * (perf.audience - 20);
        }
        thisAmount += 300 * perf.audience;
        break;
      default:
        throw new Error(`unknown type: ${play.type}`);
    }

    // add volume credits
    volumeCredits += Math.max(perf.audience - 30, 0);
    // add extra credit for every ten comedy attendees
    if ("comedy" === play.type) volumeCredits += Math.floor(perf.audience / 5);

    // print line for this order
    result += ` ${play.name}: ${format(thisAmount/100)} (${perf.audience} seats\n`;
    totalAmount += thisAmount;
  }
  result += `Amount owed is ${format(totalAmount/100)}\n`;
  result += `You earned ${volumeCredits} credits\n`;
  return result;
}
```

□□□□□□□□□□invoices.json □ plays.json□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□

```
Statement for BigCo
Hamlet: $650.00 (55 seats)
As You Like It: $580.00 (35 seats)
Othello: $500.00 (40 seats)
Amount owed is $1,730.00
You earned 47 credits
```

## 1.2 测试用例

测试用例是测试程序是否按照预期工作的方法。测试用例通常由输入、预期输出和实际输出组成。测试用例的目的是验证程序是否按照需求规格说明书中的要求工作。

测试用例通常由测试用例 ID、测试用例名称、测试用例描述、测试用例输入、测试用例预期输出和测试用例实际输出组成。测试用例的目的是验证程序是否按照需求规格说明书中的要求工作。

测试用例的目的是验证程序是否按照需求规格说明书中的要求工作。测试用例通常由输入、预期输出和实际输出组成。

### Tip

测试用例的目的是验证程序是否按照需求规格说明书中的要求工作。测试用例通常由输入、预期输出和实际输出组成。

测试用例的目的是验证程序是否按照需求规格说明书中的要求工作。测试用例通常由输入、预期输出和实际输出组成。测试用例的目的是验证程序是否按照需求规格说明书中的要求工作。

测试用例的目的是验证程序是否按照需求规格说明书中的要求工作。测试用例通常由输入、预期输出和实际输出组成。测试用例的目的是验证程序是否按照需求规格说明书中的要求工作。

测试用例的目的是验证程序是否按照需求规格说明书中的要求工作。测试用例通常由输入、预期输出和实际输出组成。测试用例的目的是验证程序是否按照需求规格说明书中的要求工作。

测试用例的目的是验证程序是否按照需求规格说明书中的要求工作。测试用例通常由输入、预期输出和实际输出组成。测试用例的目的是验证程序是否按照需求规格说明书中的要求工作。

## 1.3 测试用例

测试用例的目的是验证程序是否按照需求规格说明书中的要求工作。测试用例通常由输入、预期输出和实际输出组成。测试用例的目的是验证程序是否按照需求规格说明书中的要求工作。



```

function statement (invoice, plays) {
  let totalAmount = 0;
  let volumeCredits = 0;
  let result = `Statement for ${invoice.customer}\n`;
  const format = new Intl.NumberFormat("en-US",
    { style: "currency", currency: "USD",
      minimumFractionDigits: 2 }).format;
  for (let perf of invoice.performances) {
    const play = plays[perf.playID];
    let thisAmount = 0;

    switch (play.type) {
      case "tragedy":
        thisAmount = 40000;
        if (perf.audience > 30) {
          thisAmount += 1000 * (perf.audience - 30);
        }
        break;
      case "comedy":
        thisAmount = 30000;
        if (perf.audience > 20) {
          thisAmount += 10000 + 500 * (perf.audience - 20);
        }
        thisAmount += 300 * perf.audience;
        break;
      default:
        throw new Error(`unknown type: ${play.type}`);
    }

    // add volume credits
    volumeCredits += Math.max(perf.audience - 30, 0);
    // add extra credit for every ten comedy attendees
    if ("comedy" === play.type) volumeCredits += Math.floor(perf.audience / 5);

    // print line for this order
    result += ` ${play.name}: ${format(thisAmount/100)} (${perf.audience} seats\n`;
    totalAmount += thisAmount;
  }
  result += `Amount owed is ${format(totalAmount/100)}\n`;
  result += `You earned ${volumeCredits} credits\n`;
  return result;
}

```

Ward Cunningham  
 106 seats  
 106

amountFor(performance) 106

perf play  
 thisAmount 3  
 —thisAmount  
 106

**function statement...**



### Tip

如果你使用“ES6”的JavaScript，那么你可以使用Babel来转码你的JavaScript代码，以便在ES5的JavaScript引擎中运行。

在JavaScript中，`amountFor` 函数接收一个 `statement` 对象，并返回一个数字。

### Tip

如果你使用ES6，那么你可以使用Babel来转码你的JavaScript代码，以便在ES5的JavaScript引擎中运行。

在ES6中，你可以使用 `git` 或 `mercurial` 来管理你的代码。在ES5中，你可以使用 `push` 和 `commit` 来管理你的代码。

在ES6中，你可以使用 `IDE` 来编写你的JavaScript代码。在ES5中，你可以使用 `IDE` 来编写你的JavaScript代码。

在ES6中，你可以使用 `thisAmount` 来返回 `result`。

## function statement...

```
function amountFor(perf, play) {
  let result = 0;
  switch (play.type) {
    case "tragedy":
      result = 40000;
      if (perf.audience > 30) {
        result += 1000 * (perf.audience - 30);
      }
      break;
    case "comedy":
      result = 30000;
      if (perf.audience > 20) {
        result += 10000 + 500 * (perf.audience - 20);
      }
      result += 300 * perf.audience;
      break;
    default:
      throw new Error(`unknown type: ${play.type}`);
  }
  return result;
}
```

在ES6中，你可以使用 `result` 来返回 `result`。

## function statement...



```
function amountFor(aPerformance, play) {
  let result = 0;
  switch (play.type) {
    case "tragedy":
      result = 40000;
      if (aPerformance.audience > 30) {
        result += 1000 * (aPerformance.audience - 30);
      }
      break;
    case "comedy":
      result = 30000;
      if (aPerformance.audience > 20) {
        result += 10000 + 500 * (aPerformance.audience - 20);
      }
      result += 300 * aPerformance.audience;
      break;
    default:
      throw new Error(`unknown type: ${play.type}`);
  }
  return result;
}
```

JavaScript

Kent Beck

[Beck SBPP]

#### Tip

play

### play

amountFor aPerformance

play performance amountFor

play

178

### function statement...

```
function playFor(aPerformance) {
  return plays[aPerformance.playID];
}
```

...

```
function statement (invoice, plays) {
  let totalAmount = 0;
  let volumeCredits = 0;
  let result = `Statement for ${invoice.customer}\n`;
  const format = new Intl.NumberFormat("en-US",
    { style: "currency", currency: "USD",
      minimumFractionDigits: 2 }).format;
  for (let perf of invoice.performances) {
    const play = playFor(perf);
    let thisAmount = amountFor(perf, play);

    // add volume credits
    volumeCredits += Math.max(perf.audience - 30, 0);
    // add extra credit for every ten comedy attendees
    if ("comedy" === play.type) volumeCredits += Math.floor(perf.audience / 5);

    // print line for this order
    result += ` ${play.name}: ${format(thisAmount/100)} (${perf.audience} seats\n`;
    totalAmount += thisAmount;
  }
  result += `Amount owed is ${format(totalAmount/100)}\n`;
  result += `You earned ${volumeCredits} credits\n`;
  return result;
}
```

123 play

...

```
function statement (invoice, plays) {
  let totalAmount = 0;
  let volumeCredits = 0;
  let result = `Statement for ${invoice.customer}\n`;
  const format = new Intl.NumberFormat("en-US",
    { style: "currency", currency: "USD",
      minimumFractionDigits: 2 }).format;
  for (let perf of invoice.performances) {
    const play = playFor(perf);
    let thisAmount = amountFor(perf, playFor(perf));

    // add volume credits
    volumeCredits += Math.max(perf.audience - 30, 0);
    // add extra credit for every ten comedy attendees
    if ("comedy" === playFor(perf).type) volumeCredits += Math.floor(perf.audience / 5);

    // print line for this order
    result += ` ${playFor(perf).name}: ${format(thisAmount/100)} (${perf.audience} seats\n`;
    totalAmount += thisAmount;
  }
  result += `Amount owed is ${format(totalAmount/100)}\n`;
  result += `You earned ${volumeCredits} credits\n`;
  return result;
}
```

amountFor 124 play  
amountFor

**function statement...**

```
function amountFor(aPerformance, play) {
  let result = 0;
  switch (playFor(aPerformance).type) {
    case "tragedy":
      result = 40000;
      if (aPerformance.audience > 30) {
        result += 1000 * (aPerformance.audience - 30);
      }
      break;
    case "comedy":
      result = 30000;
      if (aPerformance.audience > 20) {
        result += 10000 + 500 * (aPerformance.audience - 20);
      }
      result += 300 * aPerformance.audience;
      break;
    default:
      throw new Error(`unknown type: ${playFor(aPerformance).type}`);
  }
  return result;
}
```

□□□□□□□□□□□□□□□□

□□□□□...

```
function statement (invoice, plays) {
  let totalAmount = 0;
  let volumeCredits = 0;
  let result = `Statement for ${invoice.customer}\n`;
  const format = new Intl.NumberFormat("en-US",
    { style: "currency", currency: "USD",
      minimumFractionDigits: 2 }).format;
  for (let perf of invoice.performances) {
    let thisAmount = amountFor(perf, playFor(perf));

    // add volume credits
    volumeCredits += Math.max(perf.audience - 30, 0);
    // add extra credit for every ten comedy attendees
    if ("comedy" === playFor(perf).type) volumeCredits += Math.floor(perf.audience / 10);

    // print line for this order
    result += ` ${playFor(perf).name}: ${format(thisAmount/100)} (${perf.audience} audience)\n`;
    totalAmount += thisAmount;
  }
  result += `Amount owed is ${format(totalAmount/100)}\n`;
  result += `You earned ${volumeCredits} credits\n`;
  return result;
}
```

**function statement...**







```
function statement (invoice, plays) {
  let totalAmount = 0;
  let volumeCredits = 0;
  let result = `Statement for ${invoice.customer}\n`;
  for (let perf of invoice.performances) {
    volumeCredits += volumeCreditsFor(perf);

    // print line for this order
    result += ` ${playFor(perf).name}: ${format(amountFor(perf)/100)} (${perf.audience})\n`;
    totalAmount += amountFor(perf);
  }
  result += `Amount owed is ${format(totalAmount/100)}\n`;
  result += `You earned ${volumeCredits} credits\n`;
  return result;
}
```

### Tip

format 函数使用 Intl.NumberFormat 类来格式化数字。format 函数是 ECMAScript 2019 新增的。format 函数使用 Intl.NumberFormat 类来格式化数字。format 函数是 ECMAScript 2019 新增的。

format 函数使用 Intl.NumberFormat 类来格式化数字。format 函数是 ECMAScript 2019 新增的。format 函数使用 Intl.NumberFormat 类来格式化数字。format 函数是 ECMAScript 2019 新增的。

format 函数...

```
function statement (invoice, plays) {
  let totalAmount = 0;
  let volumeCredits = 0;
  let result = `Statement for ${invoice.customer}\n`;
  for (let perf of invoice.performances) {
    volumeCredits += volumeCreditsFor(perf);

    // print line for this order
    result += ` ${playFor(perf).name}: ${usd(amountFor(perf))} (${perf.audience})\n`;
    totalAmount += amountFor(perf);
  }
  result += `Amount owed is ${usd(totalAmount)}\n`;
  result += `You earned ${volumeCredits} credits\n`;
  return result;
}
```

### function statement...

```
function usd(aNumber) {
  return new Intl.NumberFormat("en-US", {
    style: "currency",
    currency: "USD",
    minimumFractionDigits: 2,
  }).format(aNumber / 100);
}
```

format 函数使用 Intl.NumberFormat 类来格式化数字。format 函数是 ECMAScript 2019 新增的。format 函数使用 Intl.NumberFormat 类来格式化数字。format 函数是 ECMAScript 2019 新增的。

100 credits for the first performance and 100 credits for the second performance. The total amount is 200 credits.

100 credits

volumeCredits 100 credits for the first performance and 100 credits for the second performance. The total amount is 200 credits.

100 credits...

```
function statement (invoice, plays) {
  let totalAmount = 0;
  let volumeCredits = 0;
  let result = `Statement for ${invoice.customer}\n`;

  for (let perf of invoice.performances) {

    // print line for this order
    result += ` ${playFor(perf).name}: ${usd(amountFor(perf))} (${perf.audience})\n`;
    totalAmount += amountFor(perf);
  }
  for (let perf of invoice.performances) {
    volumeCredits += volumeCreditsFor(perf);
  }

  result += `Amount owed is ${usd(totalAmount)}\n`;
  result += `You earned ${volumeCredits} credits\n`;
  return result;
}
```

100 credits for the first performance and 100 credits for the second performance. The total amount is 200 credits.

top level...

```
function statement (invoice, plays) {
  let totalAmount = 0;
  let result = `Statement for ${invoice.customer}\n`;
  for (let perf of invoice.performances) {

    // print line for this order
    result += ` ${playFor(perf).name}: ${usd(amountFor(perf))} (${perf.audience})\n`;
    totalAmount += amountFor(perf);
  }
  let volumeCredits = 0;
  for (let perf of invoice.performances) {
    volumeCredits += volumeCreditsFor(perf);
  }
  result += `Amount owed is ${usd(totalAmount)}\n`;
  result += `You earned ${volumeCredits} credits\n`;
  return result;
}
```

volumeCredits 100 credits for the first performance and 100 credits for the second performance. The total amount is 200 credits.

function statement...





volumeCredits 4

- 227
- 223
- 106
- 123

—

totalAmount

## function statement...

```
function appleSauce() {
  let totalAmount = 0;
  for (let perf of invoice.performances) {
    totalAmount += amountFor(perf);
  }
  return totalAmount;
}
```

...

```
function statement (invoice, plays) {
  let result = `Statement for ${invoice.customer}\n`;
  for (let perf of invoice.performances) {
    result += ` ${playFor(perf).name}: ${usd(amountFor(perf))} (${perf.audience}\n`;
  }
  let totalAmount = appleSauce();

  result += `Amount owed is ${usd(totalAmount)}\n`;
  result += `You earned ${totalVolumeCredits()} credits\n`;
  return result;
}
```

totalAmount

...

```
function statement (invoice, plays) {
  let result = `Statement for ${invoice.customer}\n`;
  for (let perf of invoice.performances) {
    result += ` ${playFor(perf).name}: ${usd(amountFor(perf))} (${perf.audience}\n`;
  }
  result += `Amount owed is ${usd(totalAmount())}\n`;
  result += `You earned ${totalVolumeCredits()} credits\n`;
  return result;
}
```

## function statement...



```

function statement (invoice, plays) {
  let result = `Statement for ${invoice.customer}\n`;
  for (let perf of invoice.performances) {
    result += ` ${playFor(perf).name}: ${usd(amountFor(perf))} (${perf.audience}
  }
  result += `Amount owed is ${usd(totalAmount())}\n`;
  result += `You earned ${totalVolumeCredits()} credits\n`;
  return result;

  function totalAmount() {
    let result = 0;
    for (let perf of invoice.performances) {
      result += amountFor(perf);
    }
    return result;
  }
  function totalVolumeCredits() {
    let result = 0;
    for (let perf of invoice.performances) {
      result += volumeCreditsFor(perf);
    }
    return result;
  }
  function usd(aNumber) {
    return new Intl.NumberFormat("en-US",
      { style: "currency", currency: "USD",
        minimumFractionDigits: 2 }).format(aNumber/100);
  }
  function volumeCreditsFor(aPerformance) {
    let result = 0;
    result += Math.max(aPerformance.audience - 30, 0);
    if ("comedy" === playFor(aPerformance).type) result += Math.floor(aPerformance
    return result;
  }
  function playFor(aPerformance) {
    return plays[aPerformance.playID];
  }
  function amountFor(aPerformance) {
    let result = 0;
    switch (playFor(aPerformance).type) {
    case "tragedy":
      result = 40000;
      if (aPerformance.audience > 30) {
        result += 1000 * (aPerformance.audience - 30);
      }
      break;
    case "comedy":
      result = 30000;
      if (aPerformance.audience > 20) {
        result += 10000 + 500 * (aPerformance.audience - 20);
      }
      result += 300 * aPerformance.audience;
      break;
    default:
      throw new Error(`unknown type: ${playFor(aPerformance).type}`);
    }
    return result;
  }
}

```

statement 関数は 7 個のローカル関数を含む。この関数は、  
 7 個のローカル関数を含む。

## 1.6 関数呼び出し







```

let result = `Statement for ${data.customer}\n`;
for (let perf of data.performances) {
  result += ` ${perf.play.name}: ${usd(amountFor(perf))} (${perf.audience} seats)\n`;
}
result += `Amount owed is ${usd(totalAmount())}\n`;
result += `You earned ${totalVolumeCredits()} credits\n`;
return result;

function volumeCreditsFor(aPerformance) {
  let result = 0;
  result += Math.max(aPerformance.audience - 30, 0);
  if ("comedy" === aPerformance.play.type) result += Math.floor(aPerformance.audience / 10);
  return result;
}

function amountFor(aPerformance){
  let result = 0;
  switch (aPerformance.play.type) {
    case "tragedy":
      result = 40000;
      if (aPerformance.audience > 30) {
        result += 1000 * (aPerformance.audience - 30);
      }
      break;
    case "comedy":
      result = 30000;
      if (aPerformance.audience > 20) {
        result += 10000 + 500 * (aPerformance.audience - 20);
      }
      result += 300 * aPerformance.audience;
      break;
    default:
      throw new Error(`unknown type: ${aPerformance.play.type}`);
  }
  return result;
}

```

□□□□□□□□□□ amountFor □□□□□□□□□□□□

## function statement...

```

function enrichPerformance(aPerformance) {
  const result = Object.assign({}, aPerformance);
  result.play = playFor(result);
  result.amount = amountFor(result);
  return result;
}

function amountFor(aPerformance) {...}

```

## function renderPlainText...









```

import createStatementData from "./createStatementData.js";
function statement(invoice, plays) {
  return renderPlainText(createStatementData(invoice, plays));
}
function renderPlainText(data, plays) {
  let result = `Statement for ${data.customer}\n`;
  for (let perf of data.performances) {
    result += ` ${perf.play.name}: ${usd(perf.amount)} (${perf.audience
    } seats)\n`;
  }
  result += `Amount owed is ${usd(data.totalAmount)}\n`;
  result += `You earned ${data.totalVolumeCredits} credits\n`;
  return result;
}
function htmlStatement(invoice, plays) {
  return renderHtml(createStatementData(invoice, plays));
}
function renderHtml(data) {
  let result = `<h1>Statement for ${data.customer}</h1>\n`;
  result += "<table>\n";
  result +=
    "<tr><th>play</th><th>seats</th><th>cost</th></tr>";
  for (let perf of data.performances) {
    result += `<tr><td>${perf.play.name}</td><td>${perf.audience}</td>`;
    result += `<td>${usd(perf.amount)}</td></tr>\n`;
  }
  result += "</table>\n";
  result += `<p>Amount owed is <em>${usd(
    data.totalAmount
  )}</em></p>\n`;
  result += `<p>You earned <em>${data.totalVolumeCredits}</em> credits</p>\n`;
  return result;
}
function usd(aNumber) {
  return new Intl.NumberFormat("en-US", {
    style: "currency",
    currency: "USD",
    minimumFractionDigits: 2,
  }).format(aNumber / 100);
}

```

createStatementData.js



















```

export default function createStatementData(invoice, plays) {
  const result = {};
  result.customer = invoice.customer;
  result.performances = invoice.performances.map(enrichPerformance);
  result.totalAmount = totalAmount(result);
  result.totalVolumeCredits = totalVolumeCredits(result);
  return result;

  function enrichPerformance(aPerformance) {
    const calculator = createPerformanceCalculator(aPerformance, playFor(aPerformance));
    const result = Object.assign({}, aPerformance);
    result.play = calculator.play;
    result.amount = calculator.amount;
    result.volumeCredits = calculator.volumeCredits;
    return result;
  }

  function playFor(aPerformance) {
    return plays[aPerformance.playID]
  }

  function totalAmount(data) {
    return data.performances
      .reduce((total, p) => total + p.amount, 0);
  }

  function totalVolumeCredits(data) {
    return data.performances
      .reduce((total, p) => total + p.volumeCredits, 0);
  }
}

function createPerformanceCalculator(aPerformance, aPlay) {
  switch(aPlay.type) {
    case "tragedy": return new TragedyCalculator(aPerformance, aPlay);
    case "comedy" : return new ComedyCalculator(aPerformance, aPlay);
    default:
      throw new Error(`unknown type: ${aPlay.type}`);
  }
}

class PerformanceCalculator {
  constructor(aPerformance, aPlay) {
    this.performance = aPerformance;
    this.play = aPlay;
  }
  get amount() {
    throw new Error('subclass responsibility');
  }
  get volumeCredits() {
    return Math.max(this.performance.audience - 30, 0);
  }
}

class TragedyCalculator extends PerformanceCalculator {
  get amount() {
    let result = 40000;
    if (this.performance.audience > 30) {
      result += 1000 * (this.performance.audience - 30);
    }
    return result;
  }
}

class ComedyCalculator extends PerformanceCalculator {
  get amount() {
    let result = 30000;
    if (this.performance.audience > 20) {
      result += 10000 + 500 * (this.performance.audience - 20);
    }
    result += 300 * this.performance.audience;
    return result;
  }
  get volumeCredits() {
    return super.volumeCredits + Math.floor(this.performance.audience / 5);
  }
}

```



amountFor volumeCreditsFor createPerformanceCalculator

createStatementData

JavaScript

## 1.10

106 123 198 272

3 154

### Tip

20 Kent Beck

## 第 2 章 测试策略

测试策略是测试团队在测试过程中遵循的指导原则和标准。

### 2.1 测试策略

测试策略是测试团队在测试过程中遵循的指导原则和标准。测试策略通常包括测试目标、测试范围、测试方法、测试工具、测试环境、测试资源、测试风险、测试计划、测试报告、测试总结等。

测试策略是测试团队在测试过程中遵循的指导原则和标准。

测试策略是测试团队在测试过程中遵循的指导原则和标准。

测试策略是测试团队在测试过程中遵循的指导原则和标准。

测试策略是测试团队在测试过程中遵循的指导原则和标准。

测试策略是测试团队在测试过程中遵循的指导原则和标准。

测试策略是测试团队在测试过程中遵循的指导原则和标准。测试策略通常包括测试目标、测试范围、测试方法、测试工具、测试环境、测试资源、测试风险、测试计划、测试报告、测试总结等。

#### Tip

测试策略是测试团队在测试过程中遵循的指导原则和标准。

测试策略是测试团队在测试过程中遵循的指导原则和标准。测试策略通常包括测试目标、测试范围、测试方法、测试工具、测试环境、测试资源、测试风险、测试计划、测试报告、测试总结等。

测试策略是测试团队在测试过程中遵循的指导原则和标准。测试策略通常包括测试目标、测试范围、测试方法、测试工具、测试环境、测试资源、测试风险、测试计划、测试报告、测试总结等。

测试策略是测试团队在测试过程中遵循的指导原则和标准。测试策略通常包括测试目标、测试范围、测试方法、测试工具、测试环境、测试资源、测试风险、测试计划、测试报告、测试总结等。

### 2.2 测试策略

Kent Beck 测试策略是测试团队在测试过程中遵循的指导原则和标准。测试策略通常包括测试目标、测试范围、测试方法、测试工具、测试环境、测试资源、测试风险、测试计划、测试报告、测试总结等。

测试策略是测试团队在测试过程中遵循的指导原则和标准。测试策略通常包括测试目标、测试范围、测试方法、测试工具、测试环境、测试资源、测试风险、测试计划、测试报告、测试总结等。

□ □

## 2.3 □□□□

[illegible]

□ □ □ □ □ □ □ □ □

[illegible][illegible][illegible][illegible][illegible][illegible]

□□□□□ **bug**

[illegible]

Kent Beck  
 “  
 ”

--	--	--	--	--	--	--	--

[illegible]

bug







如果代码中使用了大量的 `if` 语句，那么代码的可读性就会降低。在编写代码时，应该尽量避免使用大量的 `if` 语句。如果必须使用 `if` 语句，那么应该尽量使用 `if-else` 语句，而不是 `if-else-if` 语句。此外，还可以使用 `switch` 语句来代替 `if` 语句。

## 避免使用大量的 if 语句

如果代码中使用了大量的 `if` 语句，那么代码的可读性就会降低。在编写代码时，应该尽量避免使用大量的 `if` 语句。如果必须使用 `if` 语句，那么应该尽量使用 `if-else` 语句，而不是 `if-else-if` 语句。此外，还可以使用 `switch` 语句来代替 `if` 语句。

### Tip

避免使用大量的 `if` 语句。

如果代码中使用了大量的 `if` 语句，那么代码的可读性就会降低。在编写代码时，应该尽量避免使用大量的 `if` 语句。如果必须使用 `if` 语句，那么应该尽量使用 `if-else` 语句，而不是 `if-else-if` 语句。此外，还可以使用 `switch` 语句来代替 `if` 语句。

### Tip

避免使用大量的 `if` 语句。

——Kent Beck

如果代码中使用了大量的 `if` 语句，那么代码的可读性就会降低。在编写代码时，应该尽量避免使用大量的 `if` 语句。如果必须使用 `if` 语句，那么应该尽量使用 `if-else` 语句，而不是 `if-else-if` 语句。此外，还可以使用 `switch` 语句来代替 `if` 语句。

如果代码中使用了大量的 `if` 语句，那么代码的可读性就会降低。在编写代码时，应该尽量避免使用大量的 `if` 语句。如果必须使用 `if` 语句，那么应该尽量使用 `if-else` 语句，而不是 `if-else-if` 语句。此外，还可以使用 `switch` 语句来代替 `if` 语句。

如果代码中使用了大量的 `if` 语句，那么代码的可读性就会降低。在编写代码时，应该尽量避免使用大量的 `if` 语句。如果必须使用 `if` 语句，那么应该尽量使用 `if-else` 语句，而不是 `if-else-if` 语句。此外，还可以使用 `switch` 语句来代替 `if` 语句。

## 避免使用大量的 if 语句

如果代码中使用了大量的 `if` 语句，那么代码的可读性就会降低。在编写代码时，应该尽量避免使用大量的 `if` 语句。如果必须使用 `if` 语句，那么应该尽量使用 `if-else` 语句，而不是 `if-else-if` 语句。此外，还可以使用 `switch` 语句来代替 `if` 语句。

如果代码中使用了大量的 `if` 语句，那么代码的可读性就会降低。在编写代码时，应该尽量避免使用大量的 `if` 语句。如果必须使用 `if` 语句，那么应该尽量使用 `if-else` 语句，而不是 `if-else-if` 语句。此外，还可以使用 `switch` 语句来代替 `if` 语句。

## 避免使用大量的 if 语句

code review 是开发过程中非常重要的一环，它可以帮助开发者发现代码中的错误，提高代码质量。在代码审查过程中，开发者需要仔细阅读代码，检查是否存在逻辑错误、拼写错误、格式问题等。同时，还需要关注代码的可读性和可维护性，确保代码结构清晰，易于理解和修改。

在代码审查过程中，开发者需要遵循一些基本原则。首先，要明确审查的目的，是为了发现错误还是为了提高代码质量。其次，要选择合适的审查人员，最好是熟悉相关模块的开发者。最后，要保持客观公正的态度，既要指出问题，也要肯定优点。

代码审查不仅是一种技术手段，更是一种团队协作的方式。通过代码审查，开发者可以相互学习，共同进步，提高整个团队的开发水平。

pull request 是代码审查的一种常见方式。开发者在完成代码修改后，可以将修改提交到代码仓库，并发起 pull request。其他开发者可以查看 pull request 中的代码变更，并进行评论和审查。这种方式可以有效地提高代码的质量和团队协作效率。

## 2.4 代码审查

“代码审查是什么？”这是一个非常基础的问题，但也是很多新手开发者容易忽略的问题。代码审查是指对代码进行仔细检查，以发现错误、提高代码质量的过程。它通常由开发人员或测试人员完成，是软件开发过程中不可或缺的一环。

代码审查可以分为静态代码审查和动态代码审查。静态代码审查是在代码运行之前进行的，主要通过人工检查或工具辅助来完成。动态代码审查则是在代码运行过程中进行的，通过测试用例来验证代码的正确性。

代码审查的好处有很多。首先，它可以及时发现代码中的错误，避免错误进入生产环境。其次，它可以提高代码的可读性和可维护性，方便后续的开发和维护。最后，它还可以促进团队成员之间的交流和合作，提高团队的开发效率。

在进行代码审查时，需要注意一些事项。首先，要明确审查的范围和目标，不要面面俱到。其次，要保持客观公正的态度，既要指出问题，也要肯定优点。最后，要及时记录和反馈审查结果，确保问题得到及时解决。

## 2.5 代码审查

代码审查是软件开发过程中非常重要的一环，它可以帮助开发者发现代码中的错误，提高代码质量。

在代码审查过程中，开发者需要仔细阅读代码，检查是否存在逻辑错误、拼写错误、格式问题等。同时，还需要关注代码的可读性和可维护性，确保代码结构清晰，易于理解和修改。

代码审查不仅是一种技术手段，更是一种团队协作的方式。通过代码审查，开发者可以相互学习，共同进步，提高整个团队的开发水平。

## 2.5 代码审查

代码审查是软件开发过程中非常重要的一环，它可以帮助开发者发现代码中的错误，提高代码质量。在代码审查过程中，开发者需要仔细阅读代码，检查是否存在逻辑错误、拼写错误、格式问题等。同时，还需要关注代码的可读性和可维护性，确保代码结构清晰，易于理解和修改。

## 124

124

### Tip

124

124

124

124

124

124

## 124

124

124

124

124





```
0000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000000000000
bug 00000000 bug 0000000000000000000000000000000000Parallel
Change00000000/expand-contract[mf-pc]00000000
```

## 2.6 YAGNI

[illegible][illegible][illegible][illegible]

□□□  
□□□  
□□□  
□□□  
□310□□□□□□□□□□□□□□□□□□□□□□□□“□□□□□□□□□□”□□□□□□□□□□□□□□  
□□□□□□□□□□□□□□□□□□□□□□□□□□□□

YAGNI[mf-yagni]——“you aren’t going to need it”YAGNI “”YAGNI “”

YAGNI

## 2.7

□□□□“□□□□□”□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□  
□□XP□[mf-xp]□□□□□□□□□□XP □□□□□□□□□□□□□□□□□□□□□□□□  
□□□□□□□□□□□□□□□□

[mf-nm]

“ ”

[illegible]







Bill Opdyke 与 Ralph 共同开发了 Smalltalk 的垃圾回收系统。Bill 在 C++ 中实现了“语义保留的 refactoring”[Opdyke]。

1992 年 OOPSLA 会议上 Bill 介绍了他的垃圾回收系统。

John Brant 与 Don Roberts 开发了 Refactoring Browser，这是 Smalltalk 的一个工具。

Kent 开发了垃圾回收系统。1992 年 OOPSLA 会议上 Kent 介绍了他的垃圾回收系统。

1992 年 OOPSLA 会议上 1 介绍了他的垃圾回收系统。Java 的垃圾回收系统是“垃圾回收”。

## 2.10 垃圾回收

10 介绍了 Java 的垃圾回收系统。IntelliJ IDEA 与 Eclipse 是 Java 的 IDE。

Smalltalk 与 Refactoring Browser 是 John Brandt 与 Don Roberts 开发的。21 介绍了 Java 的垃圾回收系统。JetBrains 与 IntelliJ IDEA 是 IDE。IBM 的 VisualAge 与 Java 的 VisualAge 是 IDE。Eclipse 是 IDE。

C# 的垃圾回收系统是 JetBrains 的 Resharper。Visual Studio 是 IDE。

垃圾回收系统是垃圾回收系统。垃圾回收系统是垃圾回收系统。

垃圾回收系统是垃圾回收系统/垃圾回收系统是垃圾回收系统。119 介绍了 Emacs 的垃圾回收系统。

垃圾回收系统是垃圾回收系统。垃圾回收系统是垃圾回收系统。IDE 是 IDE。IDE 是 IDE。

垃圾回收系统是垃圾回收系统。垃圾回收系统是垃圾回收系统。

垃圾回收系统是垃圾回收系统。124 介绍了 Salesman 的 Server。addClient 是 Salesman 的 addClient。Server 是 addClient。Smalltalk 是 addClient。



## 第 3 章 测试驱动开发

——Kent Beck 与 Martin Fowler

“测试驱动开发”

——Kent Beck 的测试驱动开发

测试驱动开发是一种软件开发方法，它要求开发者在编写代码之前，先编写测试代码。这种方法可以确保代码的正确性，并且可以及时发现错误。

测试驱动开发的核心思想是“先测试，后编码”。在编写代码之前，开发者需要先编写测试代码，以确保代码的正确性。这种方法可以确保代码的正确性，并且可以及时发现错误。

测试驱动开发 1 测试驱动开发是一种软件开发方法，它要求开发者在编写代码之前，先编写测试代码。这种方法可以确保代码的正确性，并且可以及时发现错误。

“测试”驱动开发“测试驱动开发”是一种软件开发方法，它要求开发者在编写代码之前，先编写测试代码。这种方法可以确保代码的正确性，并且可以及时发现错误。

测试驱动开发是一种软件开发方法，它要求开发者在编写代码之前，先编写测试代码。这种方法可以确保代码的正确性，并且可以及时发现错误。

测试驱动开发是一种软件开发方法，它要求开发者在编写代码之前，先编写测试代码。这种方法可以确保代码的正确性，并且可以及时发现错误。

### 3.1 测试驱动开发 Mysterious Name

测试驱动开发是一种软件开发方法，它要求开发者在编写代码之前，先编写测试代码。这种方法可以确保代码的正确性，并且可以及时发现错误。

测试驱动开发是一种软件开发方法，它要求开发者在编写代码之前，先编写测试代码。这种方法可以确保代码的正确性，并且可以及时发现错误。

测试驱动开发是一种软件开发方法，它要求开发者在编写代码之前，先编写测试代码。这种方法可以确保代码的正确性，并且可以及时发现错误。

1 测试驱动开发 International Man of Mystery 1997 测试驱动开发——测试

### 3.2 测试驱动开发 Duplicated Code

测试驱动开发是一种软件开发方法，它要求开发者在编写代码之前，先编写测试代码。这种方法可以确保代码的正确性，并且可以及时发现错误。

“”106  
223  
350

### 3.3 Long Function

“”——  
——  
——

“”“”

106

106  
178  
140319

——337

260 switch  
106 switch  
272

227

### 3.4 Long Parameter List

324  
319  
140 flag  
314

144  
 partially  
 applied function

### 3.5 Global Data

bug  
 singleton

132

132

132

### 3.6 Mutable Data

bug

132  
 240  
 223  
 106  
 API  
 306  
 331

bug  
 248

144  
 149  
 252

### 3.7 Divergent Change

144

2000 年 10 月，Google 宣布将使用 C++ 语言编写其搜索引擎的索引器。这一消息在业界引起了轰动，因为在此之前，Google 一直使用自己的专有语言编写搜索引擎。这一举措不仅展示了 Google 对 C++ 语言的信心，也反映了 C++ 语言在高性能计算领域的强大能力。

2001 年 10 月，Google 宣布将使用 C++ 语言编写其搜索引擎的索引器。这一消息在业界引起了轰动，因为在此之前，Google 一直使用自己的专有语言编写搜索引擎。这一举措不仅展示了 Google 对 C++ 语言的信心，也反映了 C++ 语言在高性能计算领域的强大能力。

### 3.8 Shotgun Surgery

2001 年 10 月，Google 宣布将使用 C++ 语言编写其搜索引擎的索引器。这一消息在业界引起了轰动，因为在此之前，Google 一直使用自己的专有语言编写搜索引擎。这一举措不仅展示了 Google 对 C++ 语言的信心，也反映了 C++ 语言在高性能计算领域的强大能力。

2001 年 10 月，Google 宣布将使用 C++ 语言编写其搜索引擎的索引器。这一消息在业界引起了轰动，因为在此之前，Google 一直使用自己的专有语言编写搜索引擎。这一举措不仅展示了 Google 对 C++ 语言的信心，也反映了 C++ 语言在高性能计算领域的强大能力。

2001 年 10 月，Google 宣布将使用 C++ 语言编写其搜索引擎的索引器。这一消息在业界引起了轰动，因为在此之前，Google 一直使用自己的专有语言编写搜索引擎。这一举措不仅展示了 Google 对 C++ 语言的信心，也反映了 C++ 语言在高性能计算领域的强大能力。

### 3.9 Feature Envy

2001 年 10 月，Google 宣布将使用 C++ 语言编写其搜索引擎的索引器。这一消息在业界引起了轰动，因为在此之前，Google 一直使用自己的专有语言编写搜索引擎。这一举措不仅展示了 Google 对 C++ 语言的信心，也反映了 C++ 语言在高性能计算领域的强大能力。

2001 年 10 月，Google 宣布将使用 C++ 语言编写其搜索引擎的索引器。这一消息在业界引起了轰动，因为在此之前，Google 一直使用自己的专有语言编写搜索引擎。这一举措不仅展示了 Google 对 C++ 语言的信心，也反映了 C++ 语言在高性能计算领域的强大能力。

2001 年 10 月，Google 宣布将使用 C++ 语言编写其搜索引擎的索引器。这一消息在业界引起了轰动，因为在此之前，Google 一直使用自己的专有语言编写搜索引擎。这一举措不仅展示了 Google 对 C++ 语言的信心，也反映了 C++ 语言在高性能计算领域的强大能力。

### 3.10 Data Clumps

2001 年 10 月，Google 宣布将使用 C++ 语言编写其搜索引擎的索引器。这一消息在业界引起了轰动，因为在此之前，Google 一直使用自己的专有语言编写搜索引擎。这一举措不仅展示了 Google 对 C++ 语言的信心，也反映了 C++ 语言在高性能计算领域的强大能力。



이것이 바로 이진 탐색의 핵심이다. 이진 탐색은 배열이 정렬되어 있을 때만 사용할 수 있다. 이진 탐색의 시간 복잡도는  $O(\log n)$ 이다.

이진 탐색은 배열의 중간 값을 찾아보는 과정을 반복한다. 이진 탐색은 배열의 크기가 작을수록 더 빠르다. 이진 탐색의 시간 복잡도는  $O(\log n)$ 이다.

이진 탐색은 배열의 중간 값을 찾아보는 과정을 반복한다. 이진 탐색은 배열의 크기가 작을수록 더 빠르다. 이진 탐색의 시간 복잡도는  $O(\log n)$ 이다.

### 3.11 원시형Primitive Obsession

이진 탐색은 배열의 중간 값을 찾아보는 과정을 반복한다. 이진 탐색은 배열의 크기가 작을수록 더 빠르다. 이진 탐색의 시간 복잡도는  $O(\log n)$ 이다.

이진 탐색은 배열의 중간 값을 찾아보는 과정을 반복한다. 이진 탐색은 배열의 크기가 작을수록 더 빠르다. 이진 탐색의 시간 복잡도는  $O(\log n)$ 이다.

이진 탐색은 배열의 중간 값을 찾아보는 과정을 반복한다. 이진 탐색은 배열의 크기가 작을수록 더 빠르다. 이진 탐색의 시간 복잡도는  $O(\log n)$ 이다.

이진 탐색은 배열의 중간 값을 찾아보는 과정을 반복한다. 이진 탐색은 배열의 크기가 작을수록 더 빠르다. 이진 탐색의 시간 복잡도는  $O(\log n)$ 이다.

### 3.12 switch switch Repeated Switches

이진 탐색은 배열의 중간 값을 찾아보는 과정을 반복한다. 이진 탐색은 배열의 크기가 작을수록 더 빠르다. 이진 탐색의 시간 복잡도는  $O(\log n)$ 이다.

이진 탐색은 배열의 중간 값을 찾아보는 과정을 반복한다. 이진 탐색은 배열의 크기가 작을수록 더 빠르다. 이진 탐색의 시간 복잡도는  $O(\log n)$ 이다.

이진 탐색은 배열의 중간 값을 찾아보는 과정을 반복한다. 이진 탐색은 배열의 크기가 작을수록 더 빠르다. 이진 탐색의 시간 복잡도는  $O(\log n)$ 이다.

### 3.13 루프Loops

이진 탐색은 배열의 중간 값을 찾아보는 과정을 반복한다. 이진 탐색은 배열의 크기가 작을수록 더 빠르다. 이진 탐색의 시간 복잡도는  $O(\log n)$ 이다.

### 3.14 게으른Lazy Element

115 186 380

### 3.15 Speculative Generality

Brian Foote

380 115 186 124

237

### 3.16 Temporary Field

182 198 289

### 3.17 Message Chains

189 106 198

192 115 399 381

### 3.18 Middle Man

192 115 399 381

198 207 189

### 3.19 Insider Trading

198 207 189

198 207 189

381 399

### 3.20 Large Class

182

depositAmount depositCurrency 375 362

5 5 10

182 375 362

182 375 362

### 3.21 Alternative Classes with Different Interfaces

124 198 375

### 3.22 Data Class

162 331

198 106

154  
 154  
 154

### 3.23 3.23 Refused Bequest

359  
 361  
 abstract

381  
 399

### 3.24 3.24 Comments

106  
 124  
 302

#### Tip

302

302

## 第 4 章 测试驱动开发

测试驱动开发（Test-Driven Development，TDD）是一种软件开发方法，它要求开发者在编写代码之前，先编写测试用例。这种方法可以帮助开发者发现代码中的错误，并确保代码的正确性。

测试驱动开发的核心思想是“测试先行”。在编写代码之前，开发者需要先编写测试用例，以确保代码能够满足需求。这种方法可以帮助开发者发现代码中的错误，并确保代码的正确性。

### 4.1 测试驱动开发

测试驱动开发（TDD）是一种软件开发方法，它要求开发者在编写代码之前，先编写测试用例。这种方法可以帮助开发者发现代码中的错误，并确保代码的正确性。TDD 的核心思想是“测试先行”。在编写代码之前，开发者需要先编写测试用例，以确保代码能够满足需求。这种方法可以帮助开发者发现代码中的错误，并确保代码的正确性。

测试“测试先行”测试驱动开发 1992 年 OOPSLA 会议上，Bedarra 和 Dave Thomas 提出了“测试先行”测试驱动开发的概念。这种方法可以帮助开发者发现代码中的错误，并确保代码的正确性。

测试驱动开发（TDD）是一种软件开发方法，它要求开发者在编写代码之前，先编写测试用例。这种方法可以帮助开发者发现代码中的错误，并确保代码的正确性。TDD 的核心思想是“测试先行”。在编写代码之前，开发者需要先编写测试用例，以确保代码能够满足需求。这种方法可以帮助开发者发现代码中的错误，并确保代码的正确性。

测试驱动开发（TDD）是一种软件开发方法，它要求开发者在编写代码之前，先编写测试用例。这种方法可以帮助开发者发现代码中的错误，并确保代码的正确性。TDD 的核心思想是“测试先行”。在编写代码之前，开发者需要先编写测试用例，以确保代码能够满足需求。这种方法可以帮助开发者发现代码中的错误，并确保代码的正确性。

#### Tip

测试驱动开发（TDD）是一种软件开发方法，它要求开发者在编写代码之前，先编写测试用例。这种方法可以帮助开发者发现代码中的错误，并确保代码的正确性。

测试驱动开发（TDD）是一种软件开发方法，它要求开发者在编写代码之前，先编写测试用例。这种方法可以帮助开发者发现代码中的错误，并确保代码的正确性。TDD 的核心思想是“测试先行”。在编写代码之前，开发者需要先编写测试用例，以确保代码能够满足需求。这种方法可以帮助开发者发现代码中的错误，并确保代码的正确性。

测试驱动开发（TDD）是一种软件开发方法，它要求开发者在编写代码之前，先编写测试用例。这种方法可以帮助开发者发现代码中的错误，并确保代码的正确性。TDD 的核心思想是“测试先行”。在编写代码之前，开发者需要先编写测试用例，以确保代码能够满足需求。这种方法可以帮助开发者发现代码中的错误，并确保代码的正确性。

#### Tip

测试驱动开发（TDD）是一种软件开发方法，它要求开发者在编写代码之前，先编写测试用例。这种方法可以帮助开发者发现代码中的错误，并确保代码的正确性。

测试驱动开发（TDD）是一种软件开发方法，它要求开发者在编写代码之前，先编写测试用例。这种方法可以帮助开发者发现代码中的错误，并确保代码的正确性。TDD 的核心思想是“测试先行”。在编写代码之前，开发者需要先编写测试用例，以确保代码能够满足需求。这种方法可以帮助开发者发现代码中的错误，并确保代码的正确性。









JavaScript `describe` 와 `it` 메서드를 사용하여 테스트를 작성할 수 있습니다. `Mocha`는 이러한 메서드를 사용하여 테스트를 실행할 수 있습니다.

다음은 `describe`와 `it` 메서드를 사용하는 예제입니다.

```
describe("province", function () {
  it("shortfall", function () {
    const asia = new Province(sampleProvinceData());
    assert.equal(asia.shortfall, 5);
  });
});
```

`Mocha`는 `describe`와 `it` 메서드를 사용하여 테스트를 작성할 수 있습니다. `it` 메서드는 `describe` 메서드와 함께 사용되며, `fixture` 메서드는 테스트 전에 실행되는 코드를 정의할 수 있습니다.

#### Tip

`describe`와 `it` 메서드는 `Mocha`에서 사용되는 메서드입니다. `describe` 메서드는 테스트의 범주를 정의하고, `it` 메서드는 테스트의 세부 사항을 정의합니다.

다음은 `NodeJS`에서 `Mocha`를 실행하는 예제입니다.

```
.....
1 passing (61ms)
```

다음은 `describe`와 `it` 메서드를 사용하여 테스트를 작성하는 예제입니다.

다음은 `describe`와 `it` 메서드를 사용하여 테스트를 작성하는 예제입니다. `bug`는 테스트 중에 발생하는 에러를 나타냅니다.

#### Tip

`describe`와 `it` 메서드는 `Mocha`에서 사용되는 메서드입니다.

## class Province...

```
get shortfall() {
  return this._demand - this.totalProduction * 2;
}
```

다음은 `describe`와 `it` 메서드를 사용하여 테스트를 작성하는 예제입니다.

```
!
0 passing (72ms)
1 failing

1) province shortfall:
AssertionError: expected -20 to equal 5
at Context.<anonymous> (src/tester.js:10:12)
```

다음은 `describe`와 `it` 메서드를 사용하여 테스트를 작성하는 예제입니다. `bug`는 테스트 중에 발생하는 에러를 나타냅니다.

assert 方法在 JavaScript 中用于断言。它通常用于测试，以确保代码按照预期工作。如果断言失败，它会抛出一个错误。

### Tip

assert 方法在 JavaScript 中用于断言。它通常用于测试，以确保代码按照预期工作。如果断言失败，它会抛出一个错误。

Mocha 是一个流行的 JavaScript 测试框架。它支持使用 Chai 断言库。Chai 提供了更丰富的断言方法，如 `assert`。

```
describe("province", function () {
  it("shortfall", function () {
    const asia = new Province(sampleProvinceData());
    assert.equal(asia.shortfall, 5);
  });
});
```

使用 `expect` 断言

```
describe("province", function () {
  it("shortfall", function () {
    const asia = new Province(sampleProvinceData());
    expect(asia.shortfall).equal(5);
  });
});
```

assert 方法在 JavaScript 中用于断言。它通常用于测试，以确保代码按照预期工作。如果断言失败，它会抛出一个错误。

Java 是一个广泛使用的编程语言。它通常与 IDE（集成开发环境）一起使用。IDE 提供了代码编辑、编译和调试的功能。在 IDE 中，你可以使用断言来测试代码。

Emacs 是一个流行的文本编辑器。它支持多种编程语言。在 Emacs 中，你可以使用断言来测试代码。

## 4.4 断言

断言是一种用于测试代码的方法。它通常用于验证代码是否符合预期。如果断言失败，它会抛出一个错误。在 Java 中，你可以使用 `public` 关键字来声明断言方法。在 Emacs 中，你可以使用 `bug` 关键字来声明断言方法。

断言是一种用于测试代码的方法。它通常用于验证代码是否符合预期。如果断言失败，它会抛出一个错误。在 Java 中，你可以使用 `public` 关键字来声明断言方法。在 Emacs 中，你可以使用 `bug` 关键字来声明断言方法。

断言是一种用于测试代码的方法。它通常用于验证代码是否符合预期。如果断言失败，它会抛出一个错误。在 Java 中，你可以使用 `public` 关键字来声明断言方法。在 Emacs 中，你可以使用 `bug` 关键字来声明断言方法。

### Tip

断言是一种用于测试代码的方法。它通常用于验证代码是否符合预期。如果断言失败，它会抛出一个错误。

```
describe("province", function () {
  it("shortfall", function () {
    const asia = new Province(sampleProvinceData());
    expect(asia.shortfall).equal(5);
  });
  it("profit", function () {
    const asia = new Province(sampleProvinceData());
    expect(asia.profit).equal(230);
  });
});
```

测试用例通过，但是测试用例中，我们使用了 `const` 来声明变量 `asia`，这导致了一个问题：在第二个测试用例中，`asia` 的值被重新赋值了，这会导致第一个测试用例的结果不正确。在 Jest 中，我们使用 `beforeEach` 来在每次测试用例之前执行代码，这可以确保每次测试用例都使用相同的初始状态。

在 Jest 中，我们使用 `beforeEach` 来在每次测试用例之前执行代码，这可以确保每次测试用例都使用相同的初始状态。

```
describe("province", function () {
  const asia = new Province(sampleProvinceData()); // DON'T DO THIS
  it("shortfall", function () {
    expect(asia.shortfall).equal(5);
  });
  it("profit", function () {
    expect(asia.profit).equal(230);
  });
});
```

测试用例通过，但是测试用例中，我们使用了 `const` 来声明变量 `asia`，这导致了一个问题：在第二个测试用例中，`asia` 的值被重新赋值了，这会导致第一个测试用例的结果不正确。在 Jest 中，我们使用 `beforeEach` 来在每次测试用例之前执行代码，这可以确保每次测试用例都使用相同的初始状态。

```
describe("province", function () {
  let asia;
  beforeEach(function () {
    asia = new Province(sampleProvinceData());
  });
  it("shortfall", function () {
    expect(asia.shortfall).equal(5);
  });
  it("profit", function () {
    expect(asia.profit).equal(230);
  });
});
```

`beforeEach` 会在每次测试用例之前执行，这可以确保每次测试用例都使用相同的初始状态。

在 Jest 中，我们使用 `beforeEach` 来在每次测试用例之前执行代码，这可以确保每次测试用例都使用相同的初始状态。

beforeEach 関数は、テストスイートの各テストの前に実行される。it 関数は、テストスイートの各テストの前に実行される。beforeEach 関数は、describe 関数の直後に実行される。

## 4.5 テストスイート

テストスイートは、テストの集合である。テストスイートは、describe 関数で定義される。

テストスイートの例として、bug に関する Producer の production に関するテストを示す。

```
describe('province'...
  it('change production', function() {
    asia.producers[0].production = 20;
    expect(asia.shortfall).equal(-6);
    expect(asia.profit).equal(292);
  });
```

beforeEach 関数は、テストスイートの各テストの前に実行される。describe 関数は、テストスイートの各テストの前に実行される。describe 関数は、setup-exercise-verify 関数、given-when-then 関数、arrange-act-assert 関数、beforeEach 関数と併用される。

### Tip

describe 関数は、テストスイートの各テストの前に実行される。describe 関数は、setup-exercise-verify 関数、given-when-then 関数、arrange-act-assert 関数、beforeEach 関数と併用される。

it 関数は、テストスイートの各テストの前に実行される。it 関数は、describe 関数の直後に実行される。

## 4.6 テストスイート

テストスイートは、テストの集合である。テストスイートは、describe 関数で定義される。

テストスイートの例として、bug に関する Producer の production に関するテストを示す。



```
describe('string for producers', function() {
  it('', function() {
    const data = {
      name: "String producers",
      producers: "",
      demand: 30,
      price: 20
    };
    const prov = new Province(data);
    expect(prov.shortfall).equal(0);
  });
});
```

테스트 실패 시 0 테스트가 실패했다는 것을 알 수 있다.

```
.....!
9 passing (74ms)
1 failing

1) string for producers :
  TypeError: doc.producers.forEach is not a function
    at new Province (src/main.js:22:19)
    at Context.<anonymous> (src/tester.js:86:18)
```

Mocha 실패 시 `failure` 메서드를 호출하여 `error` 메서드를 호출하여 “`...is not a function`”이라는 메시지를 출력한다. 이 메시지는 JavaScript 엔진에서 발생하는 오류를 나타낸다.

이 예제에서는 `producers` 속성이 빈 문자열로 설정되어 있기 때문에 `forEach` 메서드를 호출할 수 없다. JSON 데이터를 사용하여 테스트를 실행할 때는 이러한 문제를 피해야 한다.

이 예제에서는 `producers` 속성이 빈 문자열로 설정되어 있기 때문에 `forEach` 메서드를 호출할 수 없다. JSON 데이터를 사용하여 테스트를 실행할 때는 이러한 문제를 피해야 한다.

#### Tip

이 예제에서는 `producers` 속성이 빈 문자열로 설정되어 있기 때문에 `forEach` 메서드를 호출할 수 없다. JSON 데이터를 사용하여 테스트를 실행할 때는 이러한 문제를 피해야 한다.

이 예제에서는 `producers` 속성이 빈 문자열로 설정되어 있기 때문에 `forEach` 메서드를 호출할 수 없다. JSON 데이터를 사용하여 테스트를 실행할 때는 이러한 문제를 피해야 한다.

#### Tip

이 예제에서는 `producers` 속성이 빈 문자열로 설정되어 있기 때문에 `forEach` 메서드를 호출할 수 없다. JSON 데이터를 사용하여 테스트를 실행할 때는 이러한 문제를 피해야 한다.

## 4.7 테스트 실패



## 5 節 練習問題

この節では、5 節の練習問題を解くためのヒントを提供する。各問題の解答は、この節の末尾に示す。

### 5.1 練習問題

この節では、5 節の練習問題を解くためのヒントを提供する。

- 問題 1: `name` の値を `sketch` の値に設定する。この操作は、`sketch` の値を変更する。
- 問題 2: `sketch` の値を `sketch` の値に設定する。この操作は、`sketch` の値を変更する。
- 問題 3: `sketch` の値を `sketch` の値に設定する。この操作は、`sketch` の値を変更する。
- 問題 4: `sketch` の値を `sketch` の値に設定する。この操作は、`sketch` の値を変更する。
- 問題 5: `sketch` の値を `sketch` の値に設定する。この操作は、`sketch` の値を変更する。

この節では、5 節の練習問題を解くためのヒントを提供する。各問題の解答は、この節の末尾に示す。

“`sketch`” の値を `sketch` の値に設定する。この操作は、`sketch` の値を変更する。この操作は、`sketch` の値を変更する。

“`sketch`” の値を `sketch` の値に設定する。この操作は、`sketch` の値を変更する。この操作は、`sketch` の値を変更する。

この節では、5 節の練習問題を解くためのヒントを提供する。各問題の解答は、この節の末尾に示す。

“`sketch`” の値を `sketch` の値に設定する。この操作は、`sketch` の値を変更する。この操作は、`sketch` の値を変更する。

この節では、5 節の練習問題を解くためのヒントを提供する。各問題の解答は、この節の末尾に示す。

この節では、5 節の練習問題を解くためのヒントを提供する。各問題の解答は、この節の末尾に示す。

### 5.2 練習問題

この節では、5 節の練習問題を解くためのヒントを提供する。各問題の解答は、この節の末尾に示す。



1. 2023 年 1 月 1 日以前，  
 2. 2023 年 1 月 1 日以后，  
 3. 2023 年 1 月 1 日以后，

1. 2023 年 1 月 1 日以前，  
 2. 2023 年 1 月 1 日以后，  
 3. 2023 年 1 月 1 日以后，

## 第 6 章 代码重构

本章主要介绍代码重构的常用方法。

本章主要介绍代码重构的常用方法。本章主要介绍代码重构的常用方法。本章主要介绍代码重构的常用方法。——本章主要介绍代码重构的常用方法。

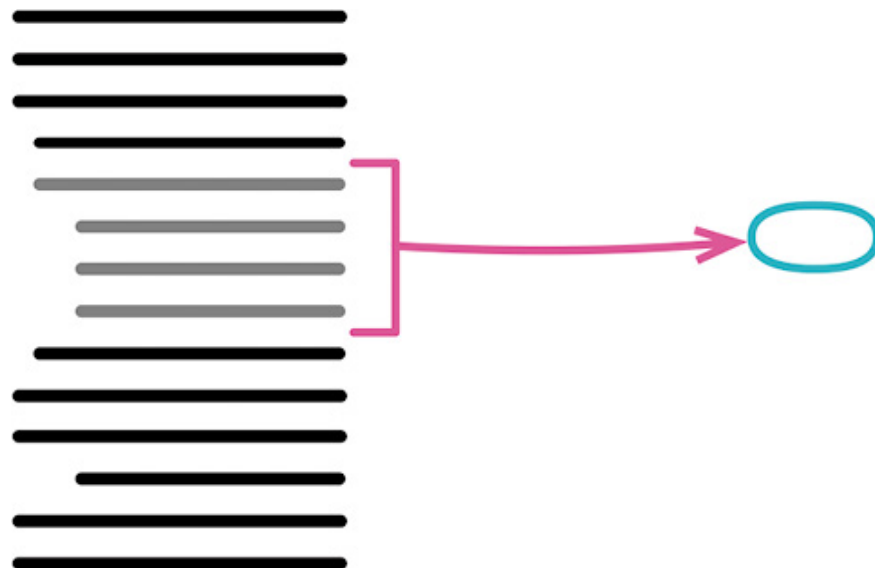
本章主要介绍代码重构的常用方法。本章主要介绍代码重构的常用方法。本章主要介绍代码重构的常用方法。本章主要介绍代码重构的常用方法。本章主要介绍代码重构的常用方法。

本章主要介绍代码重构的常用方法。本章主要介绍代码重构的常用方法。本章主要介绍代码重构的常用方法。本章主要介绍代码重构的常用方法。本章主要介绍代码重构的常用方法。

### 6.1 提取函数 Extract Function

本章主要介绍代码重构的常用方法。

本章主要介绍代码重构的常用方法。



11

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

11

```

#####“//function”#####“//method”#####
#####“//procedure”#####“//subroutine”#####
#####

```

`inline`

[illegible][illegible][illegible][illegible]

11

- [illegible]

### Tip

198



**Tip**

“ ”

```

#####
#####query#####

```

240 178

- □□□□□□□□□□□□□□

**Tip**



### Tip

□ □ □ □ □ □ □ □







```
function printOwing(invoice) {
  let outstanding = 0;

  printBanner();

  // calculate outstanding
  for (const o of invoice.orders) {
    outstanding += o.amount;
  }

  recordDueDate(invoice);
  printDetails(invoice, outstanding);
}
```

JavaScript 代码中，我们使用 `let` 来声明变量 `outstanding`，并初始化其值为 0。

然后，我们使用 `for` 循环来计算 `outstanding` 的值。

```
function printOwing(invoice) {
  printBanner();

  // calculate outstanding
  let outstanding = 0;
  for (const o of invoice.orders) {
    outstanding += o.amount;
  }

  recordDueDate(invoice);
  printDetails(invoice, outstanding);
}
```

最后，我们使用 `recordDueDate` 和 `printDetails` 函数来记录到期日期并打印详细信息。

```
function printOwing(invoice) {
  printBanner();

  // calculate outstanding
  let outstanding = 0;
  for (const o of invoice.orders) {
    outstanding += o.amount;
  }

  recordDueDate(invoice);
  printDetails(invoice, outstanding);
}

function calculateOutstanding(invoice) {
  let outstanding = 0;
  for (const o of invoice.orders) {
    outstanding += o.amount;
  }
  return outstanding;
}
```

在 `printOwing` 函数中，我们使用 `let` 来声明变量 `outstanding`，并初始化其值为 0。然后，我们使用 `for` 循环来计算 `outstanding` 的值。

JavaScript 代码中，我们使用 `let` 来声明变量 `outstanding`，并初始化其值为 0。然后，我们使用 `for` 循环来计算 `outstanding` 的值。最后，我们使用 `recordDueDate` 和 `printDetails` 函数来记录到期日期并打印详细信息。



```
function printOwing(invoice) {
  printBanner();
  let outstanding = calculateOutstanding(invoice);
  recordDueDate(invoice);
  printDetails(invoice, outstanding);
}
function calculateOutstanding(invoice) {
  let outstanding = 0;
  for (const o of invoice.orders) {
    outstanding += o.amount;
  }
  return outstanding;
}
```

このコードは、invoice オブジェクトの orders プロパティに格納された配列を反復処理し、各注文の金額を合計して、outstanding という変数に格納しています。

```
function printOwing(invoice) {
  printBanner();
  const outstanding = calculateOutstanding(invoice);
  recordDueDate(invoice);
  printDetails(invoice, outstanding);
}
function calculateOutstanding(invoice) {
  let result = 0;
  for (const o of invoice.orders) {
    result += o.amount;
  }
  return result;
}
```

このコードでは、outstanding という変数を const で宣言しています。これは、この変数の値が一度設定された後に変更されないことを保証するためです。

このコードは、invoice オブジェクトの orders プロパティに格納された配列を反復処理し、各注文の金額を合計して、outstanding という変数に格納しています。

このコードは、invoice オブジェクトの orders プロパティに格納された配列を反復処理し、各注文の金額を合計して、outstanding という変数に格納しています。このコードは、invoice オブジェクトの orders プロパティに格納された配列を反復処理し、各注文の金額を合計して、outstanding という変数に格納しています。

このコードは、invoice オブジェクトの orders プロパティに格納された配列を反復処理し、各注文の金額を合計して、outstanding という変数に格納しています。このコードは、invoice オブジェクトの orders プロパティに格納された配列を反復処理し、各注文の金額を合計して、outstanding という変数に格納しています。

## 6.2 インライン関数 Inline Function

このコードは、inline Method という関数を定義しています。

このコードは、106 という値を返します。



- 簡便な実装

簡便な実装は、コードが短く、読みやすいという利点がある。しかし、複雑なロジックを表現する際には、可読性を犠牲にする可能性がある。

例

簡便な実装の例

```
function rating(aDriver) {
  return moreThanFiveLateDeliveries(aDriver) ? 2 : 1;
}
function moreThanFiveLateDeliveries(aDriver) {
  return aDriver.numberOfLateDeliveries > 5;
}
```

簡便な実装は、return 文で結果を返すという点で簡便である。

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

簡便な実装は、関数の名前が長いという点で簡便である。

```
function rating(aDriver) {
  return moreThanFiveLateDeliveries(aDriver) ? 2 : 1;
}
function moreThanFiveLateDeliveries(dvr) {
  return dvr.numberOfLateDeliveries > 5;
}
```

簡便な実装は、moreThanFiveLateDeliveries という関数名が長いという点で簡便である。

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

簡便な実装の例

```
function reportLines(aCustomer) {
  const lines = [];
  gatherCustomerData(lines, aCustomer);
  return lines;
}
function gatherCustomerData(out, aCustomer) {
  out.push(["name", aCustomer.name]);
  out.push(["location", aCustomer.location]);
}
```















[illegible]

“ ”

```
00000000000000000000000000000000 " " rename 00000000000000000000000000000000
00000000000000000000000000000000 Rename Function 00000000000000000000000000000000
00000000000000000000
```

11

[illegible]

□ □ □ □ □

[illegible]

□□□□□□□□□□□□□□□□

□ □

111

[illegible]

□ □ □ □ □

- [illegible]

**Tip**

- [illegible]

API“”  
deprecated  
API“”

function circum(radius) {

return 2 \* Math.PI \* radius;

```
function circum(radius) {
  return 2 * Math.PI * radius;
}
```

function circumference(radius) {

```
function circumference(radius) {
  return 2 * Math.PI * radius;
}
```

circum 和 circumference

“” IDE

Person 的 changeAddress

InsuranceAgreement

function circum(radius) {

return 2 \* Math.PI \* radius;

```
function circum(radius) {
  return 2 * Math.PI * radius;
}
```

106

```
function circum(radius) {
  return circumference(radius);
}
function circumference(radius) {
  return 2 * Math.PI * radius;
}
```

115

API——API

class Book...

addReservation(customer) {  
 this.\_reservations.push(customer);  
}

class Book...

addReservation(customer) {  
 this.\_reservations.push(customer);  
}

addReservation(customer) {  
 this.\_reservations.push(customer);  
}

addReservation(customer) {  
 this.\_reservations.push(customer);  
}

class Book...

addReservation(customer) {  
 this.\_reservations.push(customer);  
}

addReservation(customer) {  
 this.\_reservations.push(customer);  
}

class Book...

addReservation(customer) {  
 this.\_reservations.push(customer);  
}

addReservation(customer) {  
 this.\_reservations.push(customer);  
}

class Book...

addReservation(customer) {  
 this.\_reservations.push(customer);  
}

addReservation(customer) {  
 this.\_reservations.push(customer);  
}

addReservation(customer) {  
 this.\_reservations.push(customer);  
}

addReservation(customer) {  
 this.\_reservations.push(customer);  
}

addReservation(customer) {  
 this.\_reservations.push(customer);  
}

customer 对象的 address 属性是一个对象，它包含 state 属性，该属性是一个字符串，表示客户所在的州。

customer 对象的 address 属性是一个对象，它包含 state 属性，该属性是一个字符串，表示客户所在的州。

```
function inNewEngland(aCustomer) {
  return ["MA", "CT", "ME", "VT", "NH", "RI"].includes(aCustomer.address.state);
}
```

customer 对象的 address 属性是一个对象，它包含 state 属性，该属性是一个字符串，表示客户所在的州。

customer 对象的 address 属性是一个对象，它包含 state 属性，该属性是一个字符串，表示客户所在的州。

```
const newEnglanders = someCustomers.filter(c => inNewEngland(c));
```

inNewEngland 函数接收一个 customer 对象作为参数，并返回一个布尔值，表示该客户是否来自新英格兰地区。该函数通过检查 customer 对象的 address 属性中的 state 属性是否包含在 ["MA", "CT", "ME", "VT", "NH", "RI"] 数组中来判断。

customer 对象的 address 属性是一个对象，它包含 state 属性，该属性是一个字符串，表示客户所在的州。

```
function inNewEngland(aCustomer) {
  const stateCode = aCustomer.address.state;
  return ["MA", "CT", "ME", "VT", "NH", "RI"].includes(stateCode);
}
```

customer 对象的 address 属性是一个对象，它包含 state 属性，该属性是一个字符串，表示客户所在的州。

```
function inNewEngland(aCustomer) {
  const stateCode = aCustomer.address.state;
  return xxNEWinNewEngland(stateCode);
}

function xxNEWinNewEngland(stateCode) {
  return ["MA", "CT", "ME", "VT", "NH", "RI"].includes(stateCode);
}
```

customer 对象的 address 属性是一个对象，它包含 state 属性，该属性是一个字符串，表示客户所在的州。

customer 对象的 address 属性是一个对象，它包含 state 属性，该属性是一个字符串，表示客户所在的州。

```
function inNewEngland(aCustomer) {
  return xxNEWinNewEngland(aCustomer.address.state);
}
```

customer 对象的 address 属性是一个对象，它包含 state 属性，该属性是一个字符串，表示客户所在的州。

customer 对象的 address 属性是一个对象，它包含 state 属性，该属性是一个字符串，表示客户所在的州。

```
const newEnglanders = someCustomers.filter(c => xxNEWinNewEngland(c.address.s
```

const newEnglanders = someCustomers.filter(c => inNewEngland(c.address.state));

const...

```
const newEnglanders = someCustomers.filter(c => inNewEngland(c.address.state));
```

function...

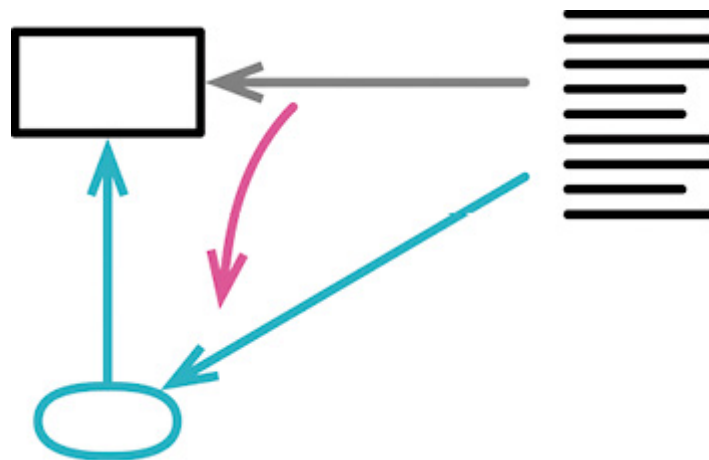
```
function inNewEngland(stateCode) {
  return ["MA", "CT", "ME", "VT", "NH", "RI"].includes(stateCode);
}
```

const newEnglanders = someCustomers.filter(c => inNewEngland(c.address.state));

## 6.6 Encapsulate Variable

Self-Encapsulate Field

Encapsulate Field



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

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

const

1. 在 `private` 方法中，我们使用 `owner` 属性来访问 `owner` 属性。  
 2. 在 `public` 方法中，我们使用 `owner` 属性来访问 `owner` 属性。  
 3. 在 `public` 方法中，我们使用 `owner` 属性来访问 `owner` 属性。

4. 在 `public` 方法中，我们使用 `owner` 属性来访问 `owner` 属性。  
 5. 在 `public` 方法中，我们使用 `owner` 属性来访问 `owner` 属性。  
 6. 在 `public` 方法中，我们使用 `owner` 属性来访问 `owner` 属性。

7. 在 `public` 方法中，我们使用 `owner` 属性来访问 `owner` 属性。  
 8. 在 `public` 方法中，我们使用 `owner` 属性来访问 `owner` 属性。

9. 在 `public` 方法中，我们使用 `owner` 属性来访问 `owner` 属性。  
 10. 在 `public` 方法中，我们使用 `owner` 属性来访问 `owner` 属性。  
 11. 在 `public` 方法中，我们使用 `owner` 属性来访问 `owner` 属性。  
 12. 在 `public` 方法中，我们使用 `owner` 属性来访问 `owner` 属性。

13. 在 `public` 方法中，我们使用 `owner` 属性来访问 `owner` 属性。  
 14. 在 `public` 方法中，我们使用 `owner` 属性来访问 `owner` 属性。  
 15. 在 `public` 方法中，我们使用 `owner` 属性来访问 `owner` 属性。  
 16. 在 `public` 方法中，我们使用 `owner` 属性来访问 `owner` 属性。

17. 在 `public` 方法中，我们使用 `owner` 属性来访问 `owner` 属性。  
 18. 在 `public` 方法中，我们使用 `owner` 属性来访问 `owner` 属性。  
 19. 在 `public` 方法中，我们使用 `owner` 属性来访问 `owner` 属性。

## 10

- 在 `private` 方法中，我们使用 `owner` 属性来访问 `owner` 属性。
- 在 `public` 方法中，我们使用 `owner` 属性来访问 `owner` 属性。
- 在 `public` 方法中，我们使用 `owner` 属性来访问 `owner` 属性。
- 在 `public` 方法中，我们使用 `owner` 属性来访问 `owner` 属性。

### Tip

在 `private` 方法中，我们使用 `owner` 属性来访问 `owner` 属性。

- 在 `private` 方法中，我们使用 `owner` 属性来访问 `owner` 属性。
- 在 `public` 方法中，我们使用 `owner` 属性来访问 `owner` 属性。

## 10

在 `private` 方法中，我们使用 `owner` 属性来访问 `owner` 属性。

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

在 `public` 方法中，我们使用 `owner` 属性来访问 `owner` 属性。

```
spaceship.owner = defaultOwner;
```

在 `public` 方法中，我们使用 `owner` 属性来访问 `owner` 属性。

```
defaultOwner = { firstName: "Rebecca", lastName: "Parsons" };
```

이제 defaultOwner 객체를 spaceship 객체에 할당합니다.

```
function getDefaultOwner() {
  return defaultOwner;
}
function setDefaultOwner(arg) {
  defaultOwner = arg;
}
```

이제 spaceship 객체의 owner 속성에 getDefaultOwner() 함수를 할당합니다.

```
spaceship.owner = getDefaultOwner();
```

이제 setDefaultOwner 함수를 호출하여 defaultOwner 객체를 설정합니다.

```
setDefaultOwner({ firstName: "Rebecca", lastName: "Parsons" });
```

이제 spaceship 객체를 console.log()로 출력합니다.

이제 spaceship 객체를 console.log()로 출력하면, spaceship 객체의 owner 속성에 getDefaultOwner() 함수가 할당되어 있고, setDefaultOwner 함수를 호출하여 defaultOwner 객체를 설정한 후, spaceship 객체를 console.log()로 출력하면, spaceship 객체의 owner 속성에 { firstName: "Rebecca", lastName: "Parsons" } 객체가 할당되어 있습니다.

## defaultOwner.js...

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

이제 defaultOwner.js 파일을 spaceship.js 파일에 import합니다. spaceship.js 파일의 맨 위에 \_\_privateOnly\_defaultOwner 변수를 선언합니다.

이제 spaceship.js 파일의 get 함수를 수정합니다.

## defaultOwner.js...

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

JavaScript에서는 객체의 속성에 함수를 할당하는 것을 "getter/setter"라고 부릅니다. "getter/setter"는 "Overloaded Getter Setter"라고도 부릅니다. (mf-orgs) 객체의 속성에 함수를 할당하는 것을 "getter/setter"라고 부릅니다.

이제



1. 在 `defaultOwner.js` 中，我们定义了一个 `defaultOwner` 函数，它返回一个默认的所有者对象。

```

const owner1 = defaultOwner();
assert.equal("Fowler", owner1.lastName, "when set");
const owner2 = defaultOwner();
owner2.lastName = "Parsons";
assert.equal("Parsons", owner1.lastName, "after change owner2"); // is this ok?
    
```

2. 在 `defaultOwner.js` 中，我们定义了一个 `setDefaultOwner` 函数，它用于设置默认的所有者数据。

3. 在 `defaultOwner.js` 中，我们定义了一个 `Person` 类，它用于创建所有者对象。

## defaultOwner.js...

```

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

4. 在 `defaultOwner.js` 中，我们定义了一个 `Person` 类，它用于创建所有者对象。

```

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

class Person {
    constructor(data) {
        this._lastName = data.lastName;
        this._firstName = data.firstName;
    }
    get lastName() {return this._lastName;}
    get firstName() {return this._firstName;}
    // and so on for other properties
}
    
```

5. 在 `defaultOwner.js` 中，我们定义了一个 `Person` 类，它用于创建所有者对象。

6. 在 `defaultOwner.js` 中，我们定义了一个 `Person` 类，它用于创建所有者对象。

7. 在 `defaultOwner.js` 中，我们定义了一个 `Person` 类，它用于创建所有者对象。

8. 在 `defaultOwner.js` 中，我们定义了一个 `Person` 类，它用于创建所有者对象。







```
const station = {
  name: "ZB1",
  readings: [
    { temp: 47, time: "2016-11-10 09:10" },
    { temp: 53, time: "2016-11-10 09:20" },
    { temp: 58, time: "2016-11-10 09:30" },
    { temp: 53, time: "2016-11-10 09:40" },
    { temp: 51, time: "2016-11-10 09:50" },
  ],
};
```

function readingsOutsideRange(station, min, max) {

```
  return station.readings
    .filter(r => r.temp < min || r.temp > max);
}
```

function readingsOutsideRange(station, min, max) {

return

```
  station,
  operatingPlan.temperatureFloor,
  operatingPlan.temperatureCeiling
);
```

function readingsOutsideRange(station, min, max) {  
 return station.readings  
 .filter(r => r.temp < min || r.temp > max);  
}

```
class NumberRange {
  constructor(min, max) {
    this._data = { min: min, max: max };
  }
  get min() {
    return this._data.min;
  }
  get max() {
    return this._data.max;
  }
}
```

JavaScript  
Value Object (mf-vo)

124 readingsOutsideRange

```
function readingsOutsideRange(station, min, max, range) {
  return station.readings
    .filter(r => r.temp < min || r.temp > max);
}
```

JavaScript null





1. The first step is to create a new function object. This is done by calling the `Function` constructor, passing the function body as a string and the context object as an argument.

2. The next step is to call the `call` method on the function object, passing the context object as an argument.

3. The final step is to return the result of the function call.

4. The function object is then returned to the caller.

5. The function object is then returned to the caller.

## Tip

- The function object is created with a prototype of `Function.prototype`.

### Tip

The function object is created with a prototype of `Function.prototype`.

- The function object is created with a prototype of `Function.prototype`.

### Tip

The function object is created with a prototype of `Function.prototype`.

- The function object is created with a prototype of `Function.prototype`.

## Tip

The function object is created with a prototype of `Function.prototype`.

```
reading = { customer: "ivan", quantity: 10, month: 5, year: 2017 };
```

The function object is created with a prototype of `Function.prototype`.

## Tip 1...

```
const aReading = acquireReading();
const baseCharge = baseRate(aReading.month, aReading.year) * aReading.quantity;
```

The function object is created with a prototype of `Function.prototype`.

## Tip 2...

```
const aReading = acquireReading();
const base = baseRate(aReading.month, aReading.year) * aReading.quantity;
const taxableCharge = Math.max(0, base - taxThreshold(aReading.year));
```





### 3...

```
const rawReading = acquireReading();
const aReading = new Reading(rawReading);
const basicChargeAmount = aReading.calculateBaseCharge;
```

124

```
get baseCharge() {
    return baseRate(this.month, this.year) * this.quantity;
}
```

### 3...

```
const rawReading = acquireReading();
const aReading = new Reading(rawReading);
const basicChargeAmount = aReading.baseCharge;
```

Reading baseCharge "Uniform Access Principle" [mf-ua]

1

### 1...

```
const rawReading = acquireReading();
const aReading = new Reading(rawReading);
const baseCharge = aReading.baseCharge;
```

123 baseCharge "baseCharge"

### 2...

```
const rawReading = acquireReading();
const aReading = new Reading(rawReading);
const taxableCharge = Math.max(
    0,
    aReading.baseCharge - taxThreshold(aReading.year)
);
```

106 taxable charge

```
function taxableChargeFn(aReading) {
    return Math.max(0, aReading.baseCharge - taxThreshold(aReading.year));
}
```

### 3...

```
const rawReading = acquireReading();
const aReading = new Reading(rawReading);
const taxableCharge = taxableChargeFn(aReading);
```



transform() method is used to transform the data into the desired format. It is a powerful tool for data manipulation and is used extensively in data science and machine learning.

transform() method is used to transform the data into the desired format. It is a powerful tool for data manipulation and is used extensively in data science and machine learning.

transform() method is used to transform the data into the desired format. It is a powerful tool for data manipulation and is used extensively in data science and machine learning.

transform() method is used to transform the data into the desired format. It is a powerful tool for data manipulation and is used extensively in data science and machine learning.

## Tip

- transform() method is used to transform the data into the desired format.

### Tip

transform() method is used to transform the data into the desired format.

- transform() method is used to transform the data into the desired format.

### Tip

transform() method is used to transform the data into the desired format.

- transform() method is used to transform the data into the desired format.
- transform() method is used to transform the data into the desired format.

## Tip

transform() method is used to transform the data into the desired format. It is a powerful tool for data manipulation and is used extensively in data science and machine learning.

```
reading = { customer: "ivan", quantity: 10, month: 5, year: 2017 };
```

transform() method is used to transform the data into the desired format.

## Tip 1...

```
const aReading = acquireReading();
const baseCharge = baseRate(aReading.month, aReading.year) * aReading.quantity;
```

transform() method is used to transform the data into the desired format.

## Tip 2...

```
const aReading = acquireReading();
const base = baseRate(aReading.month, aReading.year) * aReading.quantity;
const taxableCharge = Math.max(0, base - taxThreshold(aReading.year));
```

106

### 3...

```
const aReading = acquireReading();
const basicChargeAmount = calculateBaseCharge(aReading);

function calculateBaseCharge(aReading) {
  return baseRate(aReading.month, aReading.year) * aReading.quantity;
}
```

“”

“”

```
function enrichReading(original) {
  const result = _.cloneDeep(original);
  return result;
}
```

Lodash cloneDeep

“enrich”

enrichReading “”

### 3...

```
const rawReading = acquireReading();
const aReading = enrichReading(rawReading);
const basicChargeAmount = calculateBaseCharge(aReading);
```

198 calculateBaseCharge

```
function enrichReading(original) {
  const result = _.cloneDeep(original);
  result.baseCharge = calculateBaseCharge(result);
  return result;
}
```

aReading accumulating variable

enrichReading

### 3...

```
const rawReading = acquireReading();
const aReading = enrichReading(rawReading);
const basicChargeAmount = aReading.baseCharge;
```

calculateBaseCharge 123 baseCharge

enrichReading 106

```
it("check reading unchanged", function () {
  const baseReading = { customer: "ivan", quantity: 15, month: 5, year: 2017 };
  const oracle = _.cloneDeep(baseReading);
  enrichReading(baseReading);
  assert.deepEqual(baseReading, oracle);
});
```

1

1...

```
const rawReading = acquireReading();
const aReading = enrichReading(rawReading);
const baseCharge = aReading.baseCharge;
```

123 baseCharge

106

```
const rawReading = acquireReading();
const aReading = enrichReading(rawReading);
const base = baseRate(aReading.month, aReading.year) * aReading.quantity;
const taxableCharge = Math.max(0, base - taxThreshold(aReading.year));
```

106

```
const rawReading = acquireReading();
const aReading = enrichReading(rawReading);
const base = aReading.baseCharge;
const taxableCharge = Math.max(0, base - taxThreshold(aReading.year));
```

123 base

```
const rawReading = acquireReading();
const aReading = enrichReading(rawReading);
const taxableCharge = Math.max(
  0,
  aReading.baseCharge - taxThreshold(aReading.year)
);
```

106









```
function priceOrder(product, quantity, shippingMethod) {
  const basePrice = product.basePrice * quantity;
  const discount = Math.max(quantity - product.discountThreshold, 0)
    * product.basePrice * product.discountRate;
  const priceData = {basePrice: basePrice};
  const price = applyShipping(priceData, basePrice, shippingMethod, quantity, discount);
  return price;
}
function applyShipping(priceData, basePrice, shippingMethod, quantity, discount) {
  const shippingPerCase = (priceData.basePrice > shippingMethod.discountThreshold)
    ? shippingMethod.discountedFee : shippingMethod.feePerCase;
  const shippingCost = quantity * shippingPerCase;
  const price = priceData.basePrice - discount + shippingCost;
  return price;
}
```

shippingMethod

quantity

```
function priceOrder(product, quantity, shippingMethod) {
  const basePrice = product.basePrice * quantity;
  const discount = Math.max(quantity - product.discountThreshold, 0)
    * product.basePrice * product.discountRate;
  const priceData = {basePrice: basePrice, quantity: quantity};
  const price = applyShipping(priceData, shippingMethod, quantity, discount);
  return price;
}
function applyShipping(priceData, shippingMethod, quantity, discount) {
  const shippingPerCase = (priceData.basePrice > shippingMethod.discountThreshold)
    ? shippingMethod.discountedFee : shippingMethod.feePerCase;
  const shippingCost = priceData.quantity * shippingPerCase;
  const price = priceData.basePrice - discount + shippingCost;
  return price;
}
```

discount

```
function priceOrder(product, quantity, shippingMethod) {
  const basePrice = product.basePrice * quantity;
  const discount = Math.max(quantity - product.discountThreshold, 0)
    * product.basePrice * product.discountRate;
  const priceData = {basePrice: basePrice, quantity: quantity, discount: discount};
  const price = applyShipping(priceData, shippingMethod, discount);
  return price;
}
function applyShipping(priceData, shippingMethod, discount) {
  const shippingPerCase = (priceData.basePrice > shippingMethod.discountThreshold)
    ? shippingMethod.discountedFee : shippingMethod.feePerCase;
  const shippingCost = priceData.quantity * shippingPerCase;
  const price = priceData.basePrice - priceData.discount + shippingCost;
  return price;
}
```



## 7.0 7.0.0

162 170 174  
 178

6 144 182 186

189 192

106 195

## 7.1 Encapsulate Record

Replace Record with Data Class

```
organization = { name: "Acme Gooseberries", country: "GB" };

class Organization {
  constructor(data) {
    this._name = data.name;
    this._country = data.country;
  }
  get name() {
    return this._name;
  }
  set name(arg) {
    this._name = arg;
  }
  get country() {
    return this._country;
  }
  set country(arg) {
    this._country = arg;
  }
}
```

131

{start: 1, end: 5} {start: 1, length: 5} {end: 5, length: 5} 3 {start, end, length}

3

이러한 구조는 배열의 인덱스 3 번째 요소를 반환하는  
 배열의 인덱스 3 번째 요소를 반환하는

이러한 구조는 배열의 인덱스 3 번째 요소를 반환하는  
 이 구조는 배열의 인덱스 3 번째 요소를 반환하는  
 이 구조는 배열의 인덱스 3 번째 요소를 반환하는  
 이 구조는 배열의 인덱스 3 번째 요소를 반환하는  
 이 구조는 배열의 인덱스 3 번째 요소를 반환하는  
 이 구조는 배열의 인덱스 3 번째 요소를 반환하는  
 이 구조는 배열의 인덱스 3 번째 요소를 반환하는  
 이 구조는 배열의 인덱스 3 번째 요소를 반환하는

이러한 구조는 배열의 인덱스 3 번째 요소를 반환하는  
 이 구조는 배열의 인덱스 3 번째 요소를 반환하는

이러한 구조는 배열의 인덱스 3 번째 요소를 반환하는

이러한 구조는 배열의 인덱스 3 번째 요소를 반환하는

이러한 구조는 배열의 인덱스 3 번째 요소를 반환하는

이러한 구조는 배열의 인덱스 3 번째 요소를 반환하는  
 이 구조는 배열의 인덱스 3 번째 요소를 반환하는

이러한 구조는 배열의 인덱스 3 번째 요소를 반환하는

이러한 구조는 배열의 인덱스 3 번째 요소를 반환하는

이러한 구조는 배열의 인덱스 3 번째 요소를 반환하는  
 이 구조는 배열의 인덱스 3 번째 요소를 반환하는

이러한 구조는 배열의 인덱스 3 번째 요소를 반환하는  
 이 구조는 배열의 인덱스 3 번째 요소를 반환하는

이러한 구조는 배열의 인덱스 3 번째 요소를 반환하는

이러한 구조는 배열의 인덱스 3 번째 요소를 반환하는

이러한 구조는 배열의 인덱스 3 번째 요소를 반환하는

이러한 구조는 배열의 인덱스 3 번째 요소를 반환하는

이러한 구조는 배열의 인덱스 3 번째 요소를 반환하는

```
const organization = { name: "Acme Gooseberries", country: "GB" };
```

이러한 구조는 배열의 인덱스 3 번째 요소를 반환하는

```
result += `<h1>${organization.name}</h1>`;
organization.name = newName;
```

이러한 구조는 배열의 인덱스 3 번째 요소를 반환하는

```
function getRawDataOfOrganization() {
  return organization;
}
```



...

```
result += `<h1>${getOrganization().name}</h1>`;
```

...

```
function getRawDataOfOrganization() {
  return organization._data;
}
function getOrganization() {
  return organization;
}
```

...\_data ...

```
class Organization {
  constructor(data) {
    this._name = data.name;
    this._country = data.country;
  }
  get name() {
    return this._name;
  }
  set name(aString) {
    this._name = aString;
  }
  get country() {
    return this._country;
  }
  set country(aCountryCode) {
    this._country = aCountryCode;
  }
}
```

...

...\_data ...

...

... JSON ...

...

... ID ...

```
"1920": {
  name: "martin",
  id: "1920",
  usages: {
    "2016": {
      "1": 50,
      "2": 55,
      // remaining months of the year
    },
    "2015": {
      "1": 70,
      "2": 63,
      // remaining months of the year
    }
  },
  "38673": {
    name: "neal",
    id: "38673",
    // more customers in a similar form
```

□□□□□□□□□□□□□□□□□□□□

□□□□□...

```
customerData[customerID].usages[year][month] = amount;
```

□□□□□...

```
function compareUsage(customerID, laterYear, month) {
  const later = customerData[customerID].usages[laterYear][month];
  const earlier = customerData[customerID].usages[laterYear - 1][month];
  return { laterAmount: later, change: later - earlier };
}
```

□□□□□□□□□□□□□□□□□□□□132□□

```
function getRawDataOfCustomers() {
  return customerData;
}
function setRawDataOfCustomers(arg) {
  customerData = arg;
}
```

□□□□□...

```
getRawDataOfCustomers()[customerID].usages[year][month] = amount;
```

□□□□□...





~~~~~

```
getCustomerData().setUsage(customerID, year, month, amount);
```

## class CustomerData...

```
setUsage(customerID, year, month, amount) {
  this._data[customerID].usages[year][month] = amount;
}
```

~~~~~

~~~~~

getRawDataOfCustomers ~~~~~  
~~~~~

~~~~~

```
function getCustomerData() {
  return customerData;
}
function getRawDataOfCustomers() {
  return customerData.rawData;
}
function setRawDataOfCustomers(arg) {
  customerData = new CustomerData(arg);
}
```

## class CustomerData...

```
get rawData() {
  return _.cloneDeep(this._data);
}
```

~~~~~ lodash ~~~~~

~~~~~  
~~~~~ JavaScript ~~~~~  
~~~~~

~~~~~

~~~~~ CustomerData ~~~~

## class CustomerData...

```
usage(customerID, year, month) {
  return this._data[customerID].usages[year][month];
}
```

~~~~~

```
function compareUsage(customerID, laterYear, month) {
  const later = getCustomerData().usage(customerID, laterYear, month);
  const earlier = getCustomerData().usage(customerID, laterYear - 1, month);
  return { laterAmount: later, change: later - earlier };
}
```

customerData API 170 252 [mf-lh]

rawData

## class CustomerData...

```
get rawData() {
  return _.cloneDeep(this._data);
}
```

...

```
function compareUsage(customerID, laterYear, month) {
  const later = getCustomerData().rawData[customerID].usages[laterYear][month];
  const earlier = getCustomerData().rawData[customerID].usages[laterYear - 1][month];
  return { laterAmount: later, change: later - earlier };
}
```

Refactoring Code to Load a Document [mf-ref-doc]

customer 170 252

## 7.2 Encapsulate Collection

```
class Person {
  get courses() {return this._courses;}
  set courses(aList) {this._courses = aList;}

  class Person {
    get courses() {return this._courses.slice();}
    addCourse(aCourse) { ... }
    removeCourse(aCourse) { ... }
  }
}
```

## 问题

在代码清单 13-2 中，我们使用了一个 `Collection Pipeline` 来遍历 `aCustomer.orders`。在代码清单 13-3 中，我们使用了一个 `Collection Pipeline` 来遍历 `aCustomer.orders`。

在代码清单 13-4 中，我们使用了一个 `Collection Pipeline` 来遍历 `aCustomer.orders`。

在代码清单 13-5 中，我们使用了一个 `Collection Pipeline` 来遍历 `aCustomer.orders`。

在代码清单 13-6 中，我们使用了一个 `Collection Pipeline` 来遍历 `aCustomer.orders`。

在代码清单 13-7 中，我们使用了一个 `Collection Pipeline` 来遍历 `aCustomer.orders`。

在代码清单 13-8 中，我们使用了一个 `Collection Pipeline` 来遍历 `aCustomer.orders`。

在代码清单 13-9 中，我们使用了一个 `Collection Pipeline` 来遍历 `aCustomer.orders`。

在代码清单 13-10 中，我们使用了一个 `Collection Pipeline` 来遍历 `aCustomer.orders`。

## 问题

在代码清单 13-11 中，我们使用了一个 `Collection Pipeline` 来遍历 `aCustomer.orders`。

在代码清单 13-12 中，我们使用了一个 `Collection Pipeline` 来遍历 `aCustomer.orders`。

在代码清单 13-13 中，我们使用了一个 `Collection Pipeline` 来遍历 `aCustomer.orders`。

在代码清单 13-14 中，我们使用了一个 `Collection Pipeline` 来遍历 `aCustomer.orders`。

在代码清单 13-15 中，我们使用了一个 `Collection Pipeline` 来遍历 `aCustomer.orders`。

在代码清单 13-16 中，我们使用了一个 `Collection Pipeline` 来遍历 `aCustomer.orders`。

在代码清单 13-17 中，我们使用了一个 `Collection Pipeline` 来遍历 `aCustomer.orders`。

## 问题

在代码清单 13-18 中，我们使用了一个 `Collection Pipeline` 来遍历 `aCustomer.orders`。

**class Person...**



Person 클래스의 addCourse 메서드는 readBasicCourseNames 함수가 반환한 배열을 사용하여  
 Person 객체에 Course 객체를 추가합니다.  
 Person 클래스의 addCourse 메서드는 readBasicCourseNames 함수가 반환한 배열을 사용하여

Person 클래스...

```
for (const name of readBasicCourseNames(filename)) {
  aPerson.addCourse(new Course(name, false));
}
```

Person 클래스의 setCourse 메서드는 readBasicCourseNames 함수가 반환한 배열을 사용하여 331  
 Person 클래스의 setCourse 메서드는 readBasicCourseNames 함수가 반환한 배열을 사용하여 API

## class Person...

```
set courses(aList) {this._courses = aList.slice();}
```

Person 클래스의 setCourse 메서드는 readBasicCourseNames 함수가 반환한 배열을 사용하여  
 Person 클래스의 setCourse 메서드는 readBasicCourseNames 함수가 반환한 배열을 사용하여

## class Person...

```
get courses() {return this._courses.slice();}
```

Person 클래스의 setCourse 메서드는 readBasicCourseNames 함수가 반환한 배열을 사용하여  
 JavaScript의 sort() 메서드는 배열을 정렬합니다.  
 Person 클래스의 setCourse 메서드는 readBasicCourseNames 함수가 반환한 배열을 사용하여  
 Person 클래스의 setCourse 메서드는 readBasicCourseNames 함수가 반환한 배열을 사용하여

## 7.3 Replace Primitive with Object

Replace Data Value with Object

Replace Type Code with Class

```
orders.filter(o => "high" === o.priority
  || "rush" === o.priority);

orders.filter(o => o.priority.higherThan(new Priority("normal")))
```

Person

Person 클래스의 setCourse 메서드는 readBasicCourseNames 함수가 반환한 배열을 사용하여  
 Person 클래스의 setCourse 메서드는 readBasicCourseNames 함수가 반환한 배열을 사용하여  
 Person 클래스의 setCourse 메서드는 readBasicCourseNames 함수가 반환한 배열을 사용하여

Order 132 is a high priority order.  
 Order 132 is a high priority order.  
 Order 132 is a high priority order.  
 Order 132 is a high priority order.

Order

Order 132 is a high priority order.

Order 132 is a high priority order.

Order

Order 132 is a high priority order.

Order 132 is a high priority order.

Order

Order 124 is a high priority order.

Order 252 is a high priority order. Order 256 is a high priority order.

Order

Order 132 is a high priority order. Order 132 is a high priority order.  
 Order 132 is a high priority order.

## class Order...

```
constructor(data) {  
  this.priority = data.priority;  
  // more initialization  
}
```

Order 132 is a high priority order.

Order...

```
highPriorityCount = orders.filter(o => "high" === o.priority  
  || "rush" === o.priority)  
  .length;
```

Order 132 is a high priority order.

## class Order...

```
get priority() {return this._priority;}  
set priority(aString) {this._priority = aString;}
```

Order 132 is a high priority order.

Order 132 is a high priority order.

value class 是值类，它不能包含任何成员函数，只能包含属性。value class 是值类，它不能包含任何成员函数，只能包含属性。

```
class Priority {
    constructor(value) {
        this._value = value;
    }
    toString() {
        return this._value;
    }
}
```

toString 方法返回 value 的字符串表示。API 文档中，toString 方法返回 value 的字符串表示。

value 的字符串表示。

## class Order...

```
get priority() {return this._priority.toString();}
set priority(aString) {this._priority = new Priority(aString);}
```

Priority 类是 Order 类的属性，它不能包含任何成员函数，只能包含属性。Priority 类是 Order 类的属性，它不能包含任何成员函数，只能包含属性。

## class Order...

```
get priorityString() {return this._priority.toString();}
set priority(aString) {this._priority = new Priority(aString);}
```

...

```
highPriorityCount = orders.filter(o => "high" === o.priorityString
    || "rush" === o.priorityString)
    .length;
```

...

Priority 类是 Order 类的属性，它不能包含任何成员函数，只能包含属性。Priority 类是 Order 类的属性，它不能包含任何成员函数，只能包含属性。

## class Order...

```
get priority() {return this._priority;}
get priorityString() {return this._priority.toString();}
set priority(aString) {this._priority = new Priority(aString);}
```

...

```
highPriorityCount = orders.filter(o => "high" === o.priority.toString()
    || "rush" === o.priority.toString())
    .length;
```









```
get price() {
  const basePrice = this.basePrice;
  var discountFactor = 0.98;
  if (this.basePrice > 1000) discountFactor -= 0.03;
  return this.basePrice * discountFactor;
}
```

discountFactor 106

## class Order...

```
get price() {
  const discountFactor = this.discountFactor;
  return this.basePrice * discountFactor;
}

get discountFactor() {
  var discountFactor = 0.98;
  if (this.basePrice > 1000) discountFactor -= 0.03;
  return discountFactor;
}
```

discountFactor const

```
get price() {
  return this.basePrice * this.discountFactor;
}
```

## 7.5 Extract Class

186

```
class Person {
  get officeAreaCode() {return this._officeAreaCode;}
  get officeNumber() {return this._officeNumber;}
}

class Person {
  get officeAreaCode() {return this._telephoneNumber.areaCode;}
  get officeNumber() {return this._telephoneNumber.number;}
}

class TelephoneNumber {
  get areaCode() {return this._areaCode;}
  get number() {return this._number;}
}
```

Person 클래스의 officeAreaCode와 officeNumber 속성을  
getter와 setter 메서드로 감싸고, telephoneNumber  
속성을 getter와 setter 메서드로 감싸고, officeAreaCode와 officeNumber  
속성을 private로 선언한다.

Person 클래스의 officeAreaCode와 officeNumber 속성을  
getter와 setter 메서드로 감싸고, telephoneNumber  
속성을 getter와 setter 메서드로 감싸고, officeAreaCode와 officeNumber  
속성을 private로 선언한다.

Person

Person 클래스의 officeAreaCode와 officeNumber 속성을

getter와 setter 메서드로 감싸고, telephoneNumber

속성을 getter와 setter 메서드로 감싸고, officeAreaCode와 officeNumber

속성을 private로 선언한다. "Person" 클래스

Person 클래스의 officeAreaCode와 officeNumber 속성을 207

Person 클래스의 officeAreaCode와 officeNumber 속성을 198  
getter와 setter 메서드로 감싸고, telephoneNumber  
속성을 getter와 setter 메서드로 감싸고, officeAreaCode와 officeNumber  
속성을 private로 선언한다.

Person 클래스의 officeAreaCode와 officeNumber 속성을

getter와 setter 메서드로 감싸고, telephoneNumber 252

Person

Person 클래스의 officeAreaCode와 officeNumber 속성을

## class Person...

```
get name() {return this._name;}
set name(arg) {this._name = arg;}
get telephoneNumber() {return `${this.officeAreaCode} ${this.officeNumber}`;}
get officeAreaCode() {return this._officeAreaCode;}
set officeAreaCode(arg) {this._officeAreaCode = arg;}
get officeNumber() {return this._officeNumber;}
set officeNumber(arg) {this._officeNumber = arg;}
```

Person 클래스의 officeAreaCode와 officeNumber 속성을 TelephoneNumber 클래스의  
getter와 setter 메서드로 감싸고, telephoneNumber 속성을 getter와 setter 메서드로 감싸고, officeAreaCode와 officeNumber  
속성을 private로 선언한다.

```
class TelephoneNumber {}
```

Person 클래스의 officeAreaCode와 officeNumber 속성을 TelephoneNumber 클래스의

## class Person...

```
constructor() {
  this._telephoneNumber = new TelephoneNumber();
}
```

## class TelephoneNumber...

```
get officeAreaCode() {return this._officeAreaCode;}
set officeAreaCode(arg) {this._officeAreaCode = arg;}
```

□□□□□□□□□□207□□□□□□□□

## class Person...

```
get officeAreaCode() {return this._telephoneNumber.officeAreaCode;}
set officeAreaCode(arg) {this._telephoneNumber.officeAreaCode = arg;}
```

□□□□□□□□□□□□□□□□□□□□

## class TelephoneNumber...

```
get officeNumber() {return this._officeNumber;}
set officeNumber(arg) {this._officeNumber = arg;}
```

## class Person...

```
get officeNumber() {return this._telephoneNumber.officeNumber;}
set officeNumber(arg) {this._telephoneNumber.officeNumber = arg;}
```

□□□□□□□□□□□□□□□□□□

## class TelephoneNumber...

```
get telephoneNumber() {return `(${this.officeAreaCode}) ${this.officeNumber}`;}
```

## class Person...

```
get telephoneNumber() {return this._telephoneNumber.telephoneNumber;}
```

□□□□□□□□□□“□□□□”□□□□□□“□□□□”□office□□□□□□□□□□□□□□□□

## class TelephoneNumber...

```
get areaCode() {return this._areaCode;}
set areaCode(arg) {this._areaCode = arg;}

get number() {return this._number;}
set number(arg) {this._number = arg;}
```

## class Person...

```
get officeAreaCode() {return this._telephoneNumber.areaCode;}
set officeAreaCode(arg) {this._telephoneNumber.areaCode = arg;}
get officeNumber() {return this._telephoneNumber.number;}
set officeNumber(arg) {this._telephoneNumber.number = arg;}
```

TelephoneNumber 類別的 telephoneNumber 屬性是 TelephoneNumber 類別的实例，其 areaCode 属性是 124。

## class TelephoneNumber...

```
toString() {return `(${this.areaCode}) ${this.number}`;}
```

## class Person...

```
get telephoneNumber() {return this._telephoneNumber.toString();}
```

“Person” 类使用 TelephoneNumber 类。Person 类有一个 office 属性，该属性是 TelephoneNumber 类的实例。Value Object [mf-vo] 类是 252 页中定义的。Person 类是 182 页中定义的。

## 7.6 内联类 Inline Class

Person 类 182 页

```
class Person {
  get officeAreaCode() {return this._telephoneNumber.areaCode;}
  get officeNumber() {return this._telephoneNumber.number;}
}
class TelephoneNumber {
  get areaCode() {return this._areaCode;}
  get number() {return this._number;}
}

class Person {
  get officeAreaCode() {return this._officeAreaCode;}
  get officeNumber() {return this._officeNumber;}
}
```

Person

Person 类 182 页中定义的。Person 类有一个 office 属性，该属性是 TelephoneNumber 类的实例。Value Object [mf-vo] 类是 252 页中定义的。Person 类是 182 页中定义的。

Person 类 182 页中定义的。Person 类有一个 office 属性，该属性是 TelephoneNumber 类的实例。Value Object [mf-vo] 类是 252 页中定义的。Person 类是 182 页中定义的。

Person

Person 类 182 页中定义的。Person 类有一个 office 属性，该属性是 TelephoneNumber 类的实例。Value Object [mf-vo] 类是 252 页中定义的。Person 类是 182 页中定义的。

public 方法名(参数列表) {

// 方法体

}

...

Shipment 类中 tracking information 属性

```
class TrackingInformation {
    get shippingCompany() {
        return this._shippingCompany;
    }
    set shippingCompany(arg) {
        this._shippingCompany = arg;
    }
    get trackingNumber() {
        return this._trackingNumber;
    }
    set trackingNumber(arg) {
        this._trackingNumber = arg;
    }
    get display() {
        return `${this.shippingCompany}: ${this.trackingNumber}`;
    }
}
```

Shipment 类中 tracking information 属性

## class Shipment...

```
get trackingInfo() {
    return this._trackingInformation.display;
}
get trackingInformation() {return this._trackingInformation;}
set trackingInformation(aTrackingInformation) {
    this._trackingInformation = aTrackingInformation;
}
```

TrackingInformation 属性

Shipment 类

TrackingInformation 属性

...

```
aShipment.trackingInformation.shippingCompany = request.vendor;
```

Shipment 类中 tracking information 属性

Shipment 类中 tracking information 属性

## class Shipment...

```
set shippingCompany(arg) {this._trackingInformation.shippingCompany = arg;}
```



... ..

```
aShipment.trackingInformation.shippingCompany = request.vendor;
```

TrackingInformation 115

Shipment

display 115

## class Shipment...

```
get trackingInfo() {
    return `${this.shippingCompany}: ${this.trackingNumber}`;
}
```

“” shipping company

```
get shippingCompany() {return this._trackingInformation._shippingCompany;}
set shippingCompany(arg) {this._trackingInformation._shippingCompany = arg;}
```

207 Shipment shippingCompany

TrackingInformation

## class Shipment...

```
get trackingInfo() {
    return `${this.shippingCompany}: ${this.trackingNumber}`;
}
get shippingCompany() {return this._shippingCompany;}
set shippingCompany(arg) {this._shippingCompany = arg;}
get trackingNumber() {return this._trackingNumber;}
set trackingNumber(arg) {this._trackingNumber = arg;}
```

## 7.7 Hide Delegate

192

```
manager = aPerson.department.manager;

manager = aPerson.manager;

class Person {
    get manager() {return this.department.manager;}
}
```

“” “” “” —

Person 클래스의 constructor를 호출하여 Person 객체를 생성하고, Department 객체를 생성하여 Person 객체의 department 속성에 할당한다.

Person 클래스의 get department() 메서드를 호출하여 Department 객체를 반환한다. 이때 Department 객체는 Person 객체의 department 속성에 할당된 객체이다.

Person

Person 클래스의 constructor를 호출하여 Person 객체를 생성한다.

Person 클래스의 get department() 메서드를 호출하여 Department 객체를 반환한다.

Person 클래스의 set department(arg) 메서드를 호출하여 Department 객체를 설정한다. Delegate를 사용하여 Department 객체를 설정한다.

Person

Person

Person 클래스의 constructor를 호출하여 Person 객체를 생성하고, Department 객체를 생성하여 Person 객체의 department 속성에 할당한다.

## class Person...

```

    constructor(name) {
        this._name = name;
    }
    get name() {return this._name;}
    get department() {return this._department;}
    set department(arg) {this._department = arg;}

```

## class Department...

```

    get chargeCode() {return this._chargeCode;}
    set chargeCode(arg) {this._chargeCode = arg;}
    get manager() {return this._manager;}
    set manager(arg) {this._manager = arg;}

```

Person 클래스의 constructor를 호출하여 Person 객체를 생성하고, Department 객체를 생성하여 Person 객체의 department 속성에 할당한다.

Person 클래스의 constructor를 호출하여 Person 객체를 생성한다.

```

    manager = aPerson.department.manager;

```

Person 클래스의 constructor를 호출하여 Person 객체를 생성하고, Department 객체를 생성하여 Person 객체의 department 속성에 할당한다. 이때 Department 객체는 Person 객체의 department 속성에 할당된 객체이다.

## class Person...

```

    get manager() {return this._department.manager;}

```

Person 클래스의 constructor를 호출하여 Person 객체를 생성하고, Department 객체를 생성하여 Person 객체의 department 속성에 할당한다.

11

□ □ □ □ □ ...

```
manager = aPerson.department.manager;
```

000000 Department 0000000000000000 Person 0000000000000000 Person 00  
 department 000000

## 7.8 Remove Middle Man

□□□□□□□□□□189□

```
manager = aPerson.manager;

class Person {
    get manager() {return this.department.manager;}

    manager = aPerson.department.manager;
```

11

0000000018900“0”00000000“000000”000000000000000000000000000000000000  
 00  
 00000000081000  
 0000000000“0000000000”00000000000

[illegible]

11

□ □ □ □ □ □ □ □ □ □ □ □ □ □

[illegible]

□ □

132 115

11

Person \_\_\_\_\_  
 “ ”

□□□□□...

```
manager = aPerson.manager;
```

## class Person...



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

function foundPerson(people) {
  const candidates = ["Don", "John", "Kent"];
  return people.find(p => candidates.includes(p)) || '';
}
```

## 2.2

이 코드는 people 배열에서 Don, John, Kent 중 하나를 찾아 반환하는 함수입니다. 첫 번째 함수는 for 루프를 사용하여 배열을 순회하고, 찾으면 해당 이름을 반환합니다. 두 번째 함수는 find 메서드를 사용하여 후보 목록을 기반으로 사람을 찾습니다.

이 코드는 people 배열에서 Don, John, Kent 중 하나를 찾아 반환하는 함수입니다. 첫 번째 함수는 for 루프를 사용하여 배열을 순회하고, 찾으면 해당 이름을 반환합니다. 두 번째 함수는 find 메서드를 사용하여 후보 목록을 기반으로 사람을 찾습니다.

이 코드는 people 배열에서 Don, John, Kent 중 하나를 찾아 반환하는 함수입니다. 첫 번째 함수는 for 루프를 사용하여 배열을 순회하고, 찾으면 해당 이름을 반환합니다. 두 번째 함수는 find 메서드를 사용하여 후보 목록을 기반으로 사람을 찾습니다.

## 2.3

- people 배열에서 Don, John, Kent 중 하나를 찾아 반환하는 함수
- people 배열에서 Don, John, Kent 중 하나를 찾아 반환하는 함수
- people 배열에서 Don, John, Kent 중 하나를 찾아 반환하는 함수
- people 배열에서 Don, John, Kent 중 하나를 찾아 반환하는 함수
- people 배열에서 Don, John, Kent 중 하나를 찾아 반환하는 함수

## 8 8 8 8

198 207

213 217 223 222

227 231

237

### 8.1 Move Function

Move Method

```
class Account {
    get overdraftCharge() {...}

class AccountType {
    get overdraftCharge() {...}
```

144 182









```
function trackSummary(points) {
  const totalTime = calculateTime();
  const totalDistance = calculateDistance();
  const pace = totalTime / 60 / totalDistance ;
  return {
    time: totalTime,
    distance: totalDistance,
    pace: pace
  };

  function calculateDistance() {
    return top_calculateDistance(points);
  }
}
```

trackSummary 関数の実行結果は次のようになります。

```
{
  time: 123,
  distance: 124,
  pace: 0.008130081300813
}
```

```
function trackSummary(points) {
  const totalTime = calculateTime();
  const totalDistance = top_calculateDistance(points);
  const pace = totalTime / 60 / totalDistance;
  return {
    time: totalTime,
    distance: totalDistance,
    pace: pace,
  };
}
```

trackSummary 関数の実行結果は、totalDistance の値が 123 から 124 に変わります。これは、trackSummary 関数が、top\_calculateDistance 関数を呼び出すことで、正しい距離を計算していることを示しています。

```
function trackSummary(points) {
  const totalTime = calculateTime();
  const pace = totalTime / 60 / totalDistance(points) ;
  return {
    time: totalTime,
    distance: totalDistance(points),
    pace: pace
  };
}

function totalDistance(points) {
  let result = 0;
  for (let i = 1; i < points.length; i++) {
    result += distance(points[i-1], points[i]);
  }
  return result;
}
```

totalDistanceCache という変数を使って、totalDistance 関数の実行結果をキャッシュします。

distance 関数は、2 つの点間の距離を返します。totalDistance 関数は、この distance 関数を呼び出して、4 つの点間の距離を計算します。

```
function trackSummary(points) { ... }
function totalDistance(points) { ... }
function distance(p1,p2) { ... }
function radians(degrees) { ... }
```

distance radians totalDistance  
ES 2015 JavaScript  
Account

Account

Account

## class Account...

```
get bankCharge() {
  let result = 4.5;
  if (this._daysOverdrawn > 0) result += this.overdraftCharge;
  return result;
}

get overdraftCharge() {
  if (this.type.isPremium) {
    const baseCharge = 10;
    if (this.daysOverdrawn <= 7)
      return baseCharge;
    else
      return baseCharge + (this.daysOverdrawn - 7) * 0.85;
  }
  else
    return this.daysOverdrawn * 1.75;
}
```

account type "AccountType" overdraftCharge AccountType

overdraftCharge daysOverdrawn Account

overdraftCharge AccountType

## class AccountType...

```
overdraftCharge(daysOverdrawn) {
  if (this.isPremium) {
    const baseCharge = 10;
    if (daysOverdrawn <= 7)
      return baseCharge;
    else
      return baseCharge + (daysOverdrawn - 7) * 0.85;
  }
  else
    return daysOverdrawn * 1.75;
}
```



```

overdraftCharge(account) {
  if (this.isPremium) {
    const baseCharge = 10;
    if (account.daysOverdrawn <= 7)
      return baseCharge;
    else
      return baseCharge + (account.daysOverdrawn - 7) * 0.85;
  }
  else
    return account.daysOverdrawn * 1.75;
}

```

## 8.2 移动域 Move Field

```

class Customer {
  get plan() {return this._plan;}
  get discountRate() {return this._discountRate;}

  class Customer {
    get plan() {return this._plan;}
    get discountRate() {return this.plan.discountRate;}
  }
}

```

移动域

移动域是指将域中的成员移动到域外，或者将域外的成员移动到域中。移动域的目的是为了简化域的结构，提高代码的可读性和可维护性。

移动域的实现通常有两种方式：一种是使用域驱动设计（domain-driven design）的思想，将域中的成员移动到域外，另一种是使用域驱动设计的思想，将域外的成员移动到域中。

移动域的实现通常有两种方式：一种是使用域驱动设计（domain-driven design）的思想，将域中的成员移动到域外，另一种是使用域驱动设计的思想，将域外的成员移动到域中。

移动域的实现通常有两种方式：一种是使用域驱动设计（domain-driven design）的思想，将域中的成员移动到域外，另一种是使用域驱动设计的思想，将域外的成员移动到域中。

移动域的实现通常有两种方式：一种是使用域驱动设计（domain-driven design）的思想，将域中的成员移动到域外，另一种是使用域驱动设计的思想，将域外的成员移动到域中。

移动域的实现通常有两种方式：一种是使用域驱动设计（domain-driven design）的思想，将域中的成员移动到域外，另一种是使用域驱动设计的思想，将域外的成员移动到域中。

移动域

11

□□□□□□□□□□

100

□□□□□□

□□□□□□□□□□□□□□□□

11

□ □ □ □ □ □ □ □ □ □ □ □ □ □ □ □ □ □ □ □

302

100

□□□□□□□□

100

11

```
Customer CustomerContract
```

□“□□”□

## class Customer...

```

    constructor(name, discountRate) {
        this._name = name;
        this._discountRate = discountRate;
        this._contract = new CustomerContract(dateToday());
    }

    get discountRate() {return this._discountRate;}
    becomePreferred() {
        this._discountRate += 0.03;
        // other nice things
    }

    applyDiscount(amount) {
        return amount.subtract(amount.multiply(this._discountRate));
    }
}

```

## class CustomerContract...

```
constructor(startDate) {  
  this._startDate = startDate;  
}
```

discountRate Customer CustomerContract

discountRate 132



class Account {

private: int number; string type; double interestRate;

public: Account(int number, string type, double interestRate) {  
 this->number = number;  
 this->type = type;  
 this->interestRate = interestRate;  
}

};

Account account(162, "Savings", 0.05);

## class Account...

```
constructor(number, type, interestRate) {
  this._number = number;
  this._type = type;
  this._interestRate = interestRate;
}
get interestRate() {return this._interestRate;}
```

## class AccountType...

```
constructor(nameString) {
  this._name = nameString;
}
```

AccountType accountType = new AccountType("Savings");

AccountType accountType = new AccountType("Savings", 0.05);

## class AccountType...

```
constructor(nameString, interestRate) {
  this._name = nameString;
  this._interestRate = interestRate;
}
get interestRate() {return this._interestRate;}
```

Account account = new Account(162, "Savings", 0.05);

## class Account...





## 223

223

223

106

223

115

124

223

## 223

photo HTML

```
function renderPerson(outStream, person) {
  const result = [];
  result.push(`

${person.name}</p>`);
  result.push(renderPhoto(person.photo));
  result.push(`


```

emitPhotoData title

106 emitPhotoData

```
function photoDiv(p) {
  return ["<div>", zznew(p), "</div>"].join("\n");
}

function zznew(p) {
  return [`<p>title: ${p.title}</p>`, emitPhotoData(p)].join("\n");
}
```









```

function renderPerson(outStream, person) {
  outStream.write(`<p>${person.name}</p>\n`);
  renderPhoto(outStream, person.photo);
  zztmp(outStream, person.photo);
  outStream.write(`<p>location: ${person.photo.location}</p>\n`);
}

function listRecentPhotos(outStream, photos) {
  photos
    .filter(p => p.date > recentDateCutoff())
    .forEach(p => {
      outStream.write("<div>\n");
      zztmp(outStream, p);
      outStream.write(`<p>location: ${p.location}</p>\n`);
      outStream.write("</div>\n");
    });
}

function emitPhotoData(outStream, photo) {
  zztmp(outStream, photo);
  outStream.write(`<p>location: ${photo.location}</p>\n`);
}

function zztmp(outStream, photo) {
  outStream.write(`<p>title: ${photo.title}</p>\n`);
  outStream.write(`<p>date: ${photo.date.toDateString()}</p>\n`);
}

```

□□□□□ zztmp □□□□□□□□□□ emitPhotoData□□□□□□□□□□

```

function renderPerson(outStream, person) {
  outStream.write(`<p>${person.name}</p>\n`);
  renderPhoto(outStream, person.photo);
  emitPhotoData(outStream, person.photo);
  outStream.write(`<p>location: ${person.photo.location}</p>\n`);
}

function listRecentPhotos(outStream, photos) {
  photos
    .filter(p => p.date > recentDateCutoff())
    .forEach(p => {
      outStream.write("<div>\n");
      emitPhotoData(outStream, p);
      outStream.write(`<p>location: ${p.location}</p>\n`);
      outStream.write("</div>\n");
    });
}

function emitPhotoData(outStream, photo) {
  outStream.write(`<p>title: ${photo.title}</p>\n`);
  outStream.write(`<p>date: ${photo.date.toDateString()}</p>\n`);
}

```

## 8.5 □□□□□□□□□□□ Replace Inline Code with Function Call□

```

let appliesToMass = false;
for (const s of states) {
  if (s === "MA") appliesToMass = true;
}

appliesToMass = states.includes("MA");

```





.....  
 .....  
 .....

.....

...

.....

...

.....  
 .....  
 .....106.....  
 .....

.....  
 .....  
 .....

.....

```

1 const pricingPlan = retrievePricingPlan();
2 const order = retrieveOrder();
3 const baseCharge = pricingPlan.base;
4 let charge;
5 const chargePerUnit = pricingPlan.unit;
6 const units = order.units;
7 let discount;
8 charge = baseCharge + units * chargePerUnit;
9 let discountableUnits = Math.max(units - pricingPlan.discountThreshold, 0);
10 discount = discountableUnits * pricingPlan.discountFactor;
11 if (order.isRepeat) discount += 20;

12 charge = charge - discount;
13 chargeOrder(charge);
    
```

.....discount.....  
 7 ...let discount = 10 ...discount = ...  
 ..... discount .....  
 .....106.....

..... order ..... 2 ...const order =  
 ... ..... 6 ...const units = ...

.....  
 .....

..... 2 ..... retrieveOrder().....  
 .....  
 .....Command-Query  
 Separation[mf-cqs].....  
 .....  
 .....



이 코드는 배열의 모든 요소를 순회하고, 각 요소의 값을 2배로 곱하여 새로운 배열에 저장합니다.

예제

이 코드는 배열의 모든 요소를 순회하고, 각 요소의 값을 2배로 곱하여 새로운 배열에 저장합니다. Swap Statement (wake-swap)은 배열의 모든 요소를 순회하고, 각 요소의 값을 2배로 곱하여 새로운 배열에 저장합니다. —

이 코드는 배열의 모든 요소를 순회하고, 각 요소의 값을 2배로 곱하여 새로운 배열에 저장합니다. Swap Statement (wake-swap)은 배열의 모든 요소를 순회하고, 각 요소의 값을 2배로 곱하여 새로운 배열에 저장합니다.

## 8.7 Split Loop

```
let averageAge = 0;
let totalSalary = 0;
for (const p of people) {
  averageAge += p.age;
  totalSalary += p.salary;
}
averageAge = averageAge / people.length;

let totalSalary = 0;
for (const p of people) {
  totalSalary += p.salary;
}

let averageAge = 0;
for (const p of people) {
  averageAge += p.age;
}
averageAge = averageAge / people.length;
```

이 코드는 배열의 모든 요소를 순회하고, 각 요소의 값을 2배로 곱하여 새로운 배열에 저장합니다.

이 코드는 배열의 모든 요소를 순회하고, 각 요소의 값을 2배로 곱하여 새로운 배열에 저장합니다. Swap Statement (wake-swap)은 배열의 모든 요소를 순회하고, 각 요소의 값을 2배로 곱하여 새로운 배열에 저장합니다.

이 코드는 배열의 모든 요소를 순회하고, 각 요소의 값을 2배로 곱하여 새로운 배열에 저장합니다. Swap Statement (wake-swap)은 배열의 모든 요소를 순회하고, 각 요소의 값을 2배로 곱하여 새로운 배열에 저장합니다. 106

이 코드는 배열의 모든 요소를 순회하고, 각 요소의 값을 2배로 곱하여 새로운 배열에 저장합니다. Swap Statement (wake-swap)은 배열의 모든 요소를 순회하고, 각 요소의 값을 2배로 곱하여 새로운 배열에 저장합니다. 2.8

이 코드는 배열의 모든 요소를 순회하고, 각 요소의 값을 2배로 곱하여 새로운 배열에 저장합니다.

이 코드는 배열의 모든 요소를 순회하고, 각 요소의 값을 2배로 곱하여 새로운 배열에 저장합니다.

이 코드는 배열의 모든 요소를 순회하고, 각 요소의 값을 2배로 곱하여 새로운 배열에 저장합니다.

이 코드는 배열의 모든 요소를 순회하고, 각 요소의 값을 2배로 곱하여 새로운 배열에 저장합니다.

이 코드는 배열의 모든 요소를 순회하고, 각 요소의 값을 2배로 곱하여 새로운 배열에 저장합니다. 106

## 11

total salary youngest

```
let youngest = people[0] ? people[0].age : Infinity;
let totalSalary = 0;
for (const p of people) {
  if (p.age < youngest) youngest = p.age;
  totalSalary += p.salary;
}

return `youngestAge: ${youngest}, totalSalary: ${totalSalary}`;
```

```
let youngest = people[0] ? people[0].age : Infinity;
let totalSalary = 0;
for (const p of people) {
  if (p.age < youngest) youngest = p.age;
  totalSalary += p.salary;
}

for (const p of people) {
  if (p.age < youngest) youngest = p.age;
  totalSalary += p.salary;
}

return `youngestAge: ${youngest}, totalSalary: ${totalSalary}`;
```

```
let youngest = people[0] ? people[0].age : Infinity;
let totalSalary = 0;
for (const p of people) {
  if (p.age < youngest) youngest = p.age;
  totalSalary += p.salary;
}

for (const p of people) {
  if (p.age < youngest) youngest = p.age;
  totalSalary += p.salary;
}

return `youngestAge: ${youngest}, totalSalary: ${totalSalary}`;
```

223

```
let totalSalary = 0;
for (const p of people) {
  totalSalary += p.salary;
}

let youngest = people[0] ? people[0].age : Infinity;
for (const p of people) {
  if (p.age < youngest) youngest = p.age;
}

return `youngestAge: ${youngest}, totalSalary: ${totalSalary}`;
```

106

```
return `youngestAge: ${youngestAge()}, totalSalary: ${totalSalary()}`;

function totalSalary() {
  let totalSalary = 0;
  for (const p of people) {
    totalSalary += p.salary;
  }
  return totalSalary;
}

function youngestAge() {
  let youngest = people[0] ? people[0].age : Infinity;
  for (const p of people) {
    if (p.age < youngest) youngest = p.age;
  }
  return youngest;
}
```

totalSalary 231  
youngestAge 195

```
return `youngestAge: ${youngestAge()}, totalSalary: ${totalSalary()}`;

function totalSalary() {
  return people.reduce((total, p) => total + p.salary, 0);
}

function youngestAge() {
  return Math.min(...people.map(p => p.age));
}
```

## 8.8 Replace Loop with Pipeline

```
const names = [];
for (const i of input) {
  if (i.job === "programmer")
    names.push(i.name);
}

const names = input
  .filter(i => i.job === "programmer")
  .map(i => i.name);
```

11

collection pipeline (mf-cp) map filter map filter

11

office, country, telephone

Chicago, USA, +1 312 373 1000

Beijing, China, +86 4008 900 505

Bangalore, India, +91 80 4064 9570

Porto Alegre, Brazil, +55 51 3079 3550

Chennai, India, +91 44 660 44766

... (more data follows)

function acquireData(input) {  
 const lines = input.split("\n");  
 let firstLine = true;  
 const result = [];  
 for (const line of lines) {  
 if (firstLine) {  
 firstLine = false;  
 continue;  
 }  
 if (line.trim() === "") continue;  
 const record = line.split(",");  
 if (record[1].trim() === "India") {  
 result.push({ city: record[0].trim(), phone: record[2].trim() });  
 }  
 }  
 return result;  
}

acquireData office city  
telephone number

function acquireData(input) {  
 const lines = input.split("\n");  
 let firstLine = true;  
 const result = [];  
 for (const line of lines) {  
 if (firstLine) {  
 firstLine = false;  
 continue;  
 }  
 if (line.trim() === "") continue;  
 const record = line.split(",");  
 if (record[1].trim() === "India") {  
 result.push({ city: record[0].trim(), phone: record[2].trim() });  
 }  
 }  
 return result;  
}

office, country, telephone

Chicago, USA, +1 312 373 1000



map 함수를 사용하여 record 객체를 생성합니다.

map 함수를 사용하여 record 객체를 생성합니다. record 객체는 city와 phone 속성을 가집니다.

```
function acquireData(input) {
  const lines = input.split("\n");
  const result = [];
  const loopItems = lines
    .slice(1)
    .filter(line => line.trim() !== "")
    .map(line => line.split(","));
  for (const line of loopItems) {
    const record = line;
    if (record[1].trim() === "India") {
      result.push({city: record[0].trim(), phone: record[2].trim()});
    }
  }
  return result;
}
```

filter 함수를 사용하여 record 객체를 생성합니다.

```
function acquireData(input) {
  const lines = input.split("\n");
  const result = [];
  const loopItems = lines
    .slice(1)
    .filter(line => line.trim() !== "")
    .map(line => line.split(","))
    .filter(record => record[1].trim() === "India");
  for (const line of loopItems) {
    const record = line;
    if (record[1].trim() === "India") {
      result.push({city: record[0].trim(), phone: record[2].trim()});
    }
  }
  return result;
}
```

map 함수를 사용하여 record 객체를 생성합니다.

```
function acquireData(input) {
  const lines = input.split("\n");
  const result = [];
  const loopItems = lines
    .slice(1)
    .filter(line => line.trim() !== "")
    .map(line => line.split(","))
    .filter(record => record[1].trim() === "India")
    .map(record => ({city: record[0].trim(), phone: record[2].trim()}));
  for (const line of loopItems) {
    const record = line;
    result.push(record);
  }
  return result;
}
```

map 함수를 사용하여 record 객체를 생성합니다.



```
function acquireData(input) {
  const lines = input.split("\n");
  const result = lines
    .slice(1)
    .filter(line => line.trim() !== "")
    .map(line => line.split(","))
    .filter(record => record[1].trim() === "India")
    .map(record => ({city: record[0].trim(), phone: record[2].trim()}))
    ;
  for (const line of loopItems) {
    const record = line;
    result.push(record);
  }
  return result;
}
```

이 코드는 lines 배열을 result 배열에 push하는 방식으로 result 배열을 반환한다.

```
function acquireData(input) {
  const lines = input.split("\n");
  return lines
    .slice(1)
    .filter(line => line.trim() !== "")
    .map(line => line.split(","))
    .filter(fields => fields[1].trim() === "India")
    .map(fields => ({city: fields[0].trim(), phone: fields[2].trim()}))
    ;
}
```

이 코드는 lines 배열을 반환하는 방식으로 result 배열을 반환한다.

이 코드는

“Refactoring with Loops and Collection Pipelines”[mf-ref-pipe]

## 8.9 Remove Dead Code

```
if (false) {
  doSomethingThatUsedToMatter();
}
```

이 코드는

이 코드는 doSomethingThatUsedToMatter() 함수를 호출하는 코드이지만, 이 함수는 실제로 호출되지 않는다. 따라서 이 코드는 Dead Code이다.

이 코드는 if (false) { ... } 구조를 사용하여 Dead Code를 제거하는 방법이다. 이 코드는 실제로 호출되지 않는다. 따라서 이 코드는 Dead Code이다.

이 코드는 if (false) { ... } 구조를 사용하여 Dead Code를 제거하는 방법이다. 이 코드는 실제로 호출되지 않는다. 따라서 이 코드는 Dead Code이다.



## 9.1 Split Variable

Remove Assignments to Parameters

Split Temp

```
let temp = 2 * (height + width);
console.log(temp);
temp = height * width;
console.log(temp);

const perimeter = 2 * (height + width);
console.log(perimeter);
const area = height * width;
console.log(area);
```

### 9.1 Split Variable

Remove Assignments to Parameters

Split Temp

```
let temp = 2 * (height + width);
console.log(temp);
temp = height * width;
console.log(temp);

const perimeter = 2 * (height + width);
console.log(perimeter);
const area = height * width;
console.log(area);
```

for

loop variable

collecting variable

for

for

for

for

for

for

for

for

for

function distanceTravelled (scenario, time) {  
 let result;  
 let acc = scenario.primaryForce / scenario.mass;  
 let primaryTime = Math.min(time, scenario.delay);  
 result = 0.5 \* acc \* primaryTime \* primaryTime;  
 let secondaryTime = time - scenario.delay;  
 if (secondaryTime > 0) {  
 let primaryVelocity = acc \* scenario.delay;  
 acc = (scenario.primaryForce + scenario.secondaryForce) / scenario.mass;  
 result += primaryVelocity \* secondaryTime + 0.5 \* acc \* secondaryTime \* secondaryTime;  
 }  
 return result;  
}

acc 是 scenario.primaryForce / scenario.mass  
acc 是 (scenario.primaryForce + scenario.secondaryForce) / scenario.mass

symbol 是 scenario.delay  
symbol 是 scenario.delay

const 是 scenario.delay  
acc 是 scenario.delay

```
function distanceTravelled (scenario, time) {
  let result;
  const primaryAcceleration = scenario.primaryForce / scenario.mass;
  let primaryTime = Math.min(time, scenario.delay);
  result = 0.5 * primaryAcceleration * primaryTime * primaryTime;
  let secondaryTime = time - scenario.delay;
  if (secondaryTime > 0) {
    let primaryVelocity = primaryAcceleration * scenario.delay;
    let acc = (scenario.primaryForce + scenario.secondaryForce) / scenario.mass;
    result += primaryVelocity * secondaryTime + 0.5 * acc * secondaryTime * secondaryTime;
  }
  return result;
}
```

acc 是 scenario.primaryForce / scenario.mass  
acc 是 (scenario.primaryForce + scenario.secondaryForce) / scenario.mass

acc 是 scenario.delay  
acc 是 scenario.delay



## 9.2 重命名 Rename Field

```
class Organization {
  get name() {...}
}

class Organization {
  get title() {...}
}
```

11

重命名操作通常用于将旧名称替换为新名称。例如，将 `name` 重命名为 `title`。在以下代码中，我们展示了如何重命名 `name` 属性。

在以下代码中，我们展示了如何重命名 `name` 属性。

在以下代码中，我们展示了如何重命名 `name` 属性。

11

在以下代码中，我们展示了如何重命名 `name` 属性。

在以下代码中，我们展示了如何重命名 `name` 属性。

在以下代码中，我们展示了如何重命名 `name` 属性。

11

在以下代码中，我们展示了如何重命名 `name` 属性。

在以下代码中，我们展示了如何重命名 `name` 属性。

在以下代码中，我们展示了如何重命名 `name` 属性。

在以下代码中，我们展示了如何重命名 `name` 属性。

```
const organization = { name: "Acme Gooseberries", country: "GB" };
```

在以下代码中，我们展示了如何重命名 `name` 属性。













```

get production() {
  assert(this._productionAccumulator === this.calculatedProductionAccumulator);
  return this._initialProduction + this._productionAccumulator;
}

get calculatedProductionAccumulator() {
  return this._adjustments
    .reduce((sum, a) => sum + a.amount, 0);
}

```

calculatedProduction Accumulator

## 9.4 Change Reference to Value

256

```

class Product {
  applyDiscount(arg) {this._price.amount -= arg;}

  class Product {
    applyDiscount(arg) {
      this._price = new Money(this._price.amount - arg, this._price.currency);
    }
  }
}

```

11

[mf-vo]

11

331

11

Person Telephone Number



```

    get officeAreaCode() {return this._telephoneNumber.areaCode;}
    set officeAreaCode(arg) {
        this._telephoneNumber = new TelephoneNumber(arg, this.officeNumber);
    }
    get officeNumber() {return this._telephoneNumber.number;}
    set officeNumber(arg) {
        this._telephoneNumber = new TelephoneNumber(this.officeAreaCode, arg);
    }

```

TelephoneNumber 객체와 TelephoneNumber 객체를 비교하는 equals 메서드를 구현합니다. JavaScript 객체와 비교하는 "===" 연산자를 사용합니다. equals 메서드는

## class TelephoneNumber...

```

    equals(other) {
        if (!(other instanceof TelephoneNumber)) return false;
        return this.areaCode === other.areaCode && this.number === other.number;
    }

```

테스트 코드를 작성합니다.

```

it("telephone equals", function () {
    assert(
        new TelephoneNumber("312", "555-0142").equals(
            new TelephoneNumber("312", "555-0142")
        )
    );
});

```

이제 TelephoneNumber 클래스를 완성합니다.

이제 TelephoneNumber 클래스를 완성합니다.

이제 TelephoneNumber 클래스를 완성합니다. Ruby에서는 == 연산자를 사용합니다. Java에서는 Object.equals() 메서드를 사용합니다. Java에서는 Object.hashCode() 메서드를 사용합니다.

이제 TelephoneNumber 클래스를 완성합니다. 331번 문제를 해결합니다. null 값을 처리합니다.

## 9.5 Change Value to Reference

이제 252번 문제를 해결합니다.

```

let customer = new Customer(customerData);

let customer = customerRepository.get(customerData.id);

```

이제

Order 객체를 생성할 때, data 객체를 전달합니다. data 객체는 number, customer, 그리고 다른 데이터를 포함합니다. 이 객체를 사용하여 Order 객체를 생성합니다.

Order 객체는 number, customer, 그리고 다른 데이터를 포함합니다. 이 객체를 사용하여 Order 객체를 생성합니다.

Order 객체는 number, customer, 그리고 다른 데이터를 포함합니다. 이 객체를 사용하여 Order 객체를 생성합니다.

Order 객체는 number, customer, 그리고 다른 데이터를 포함합니다. 이 객체를 사용하여 Order 객체를 생성합니다.

## Order 객체

Order 객체는 number, customer, 그리고 다른 데이터를 포함합니다.

Order 객체는 number, customer, 그리고 다른 데이터를 포함합니다.

Order 객체는 number, customer, 그리고 다른 데이터를 포함합니다.

## Order 객체

Order 객체는 number, customer, 그리고 다른 데이터를 포함합니다. JSON 객체를 사용하여 Order 객체를 생성합니다. customer ID를 사용하여 Customer 객체를 생성합니다.

## class Order...

```
constructor(data) {
  this._number = data.number;
  this._customer = new Customer(data.customer);
  // load other data
}
get customer() {return this._customer;}
```

## class Customer...

```
constructor(id) {
  this._id = id;
}
get id() {return this._id;}
```

Customer 객체는 ID를 사용하여 생성됩니다. ID는 123입니다. Customer 객체는 number, customer, 그리고 다른 데이터를 포함합니다. Customer 객체는 number, customer, 그리고 다른 데이터를 포함합니다. Customer 객체는 number, customer, 그리고 다른 데이터를 포함합니다.

Customer 객체는 number, customer, 그리고 다른 데이터를 포함합니다. Customer 객체는 number, customer, 그리고 다른 데이터를 포함합니다. Customer 객체는 number, customer, 그리고 다른 데이터를 포함합니다.







□□□□□□□□□□□□□□□□

```

if (summer())
  charge = quantity * plan.summerRate;
else
  charge = quantity * plan.regularRate + plan.regularServiceCharge;

function summer() {
  return !aDate.isBefore(plan.summerStart) && !aDate.isAfter(plan.summerEnd);
}

```

□□□□□□□□□□□□

```

if (summer())
  charge = summerCharge();
else
  charge = quantity * plan.regularRate + plan.regularServiceCharge;

function summer() {
  return !aDate.isBefore(plan.summerStart) && !aDate.isAfter(plan.summerEnd);
}
function summerCharge() {
  return quantity * plan.summerRate;
}

```

□□□□□□□□□□□□

```

if (summer())
  charge = summerCharge();
else
  charge = regularCharge();

function summer() {
  return !aDate.isBefore(plan.summerStart) && !aDate.isAfter(plan.summerEnd);
}
function summerCharge() {
  return quantity * plan.summerRate;
}
function regularCharge() {
  return quantity * plan.regularRate + plan.regularServiceCharge;
}

```

□□□□□□□□□□□□□□□□□□□□

```

charge = summer() ? summerCharge() : regularCharge();

function summer() {
  return !aDate.isBefore(plan.summerStart) && !aDate.isAfter(plan.summerEnd);
}
function summerCharge() {
  return quantity * plan.summerRate;
}
function regularCharge() {
  return quantity * plan.regularRate + plan.regularServiceCharge;
}

```

## 10.2 条件表达式 Consolidate Conditional Expression

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

if (isNotEligibleForDisability()) return 0;

function isNotEligibleForDisability() {
    return ((anEmployee.seniority < 2)
        || (anEmployee.monthsDisabled > 12)
        || (anEmployee.isPartTime));
}
```

10

如果员工的 seniority 小于 2，或者 monthsDisabled 大于 12，或者 isPartTime 为 true，那么该员工就不符合残疾津贴的条件。

如果员工不符合残疾津贴的条件，那么该员工就不符合残疾津贴的条件。如果员工符合残疾津贴的条件，那么该员工就可以获得 106 美元的残疾津贴。

如果员工符合残疾津贴的条件，那么该员工就可以获得 106 美元的残疾津贴。

10

如果员工符合残疾津贴的条件，那么该员工就可以获得 106 美元的残疾津贴。

如果员工符合残疾津贴的条件，那么该员工就可以获得 106 美元的残疾津贴。

如果员工符合残疾津贴的条件，那么该员工就可以获得 106 美元的残疾津贴。

如果员工符合残疾津贴的条件，那么该员工就可以获得 106 美元的残疾津贴。

10

如果员工符合残疾津贴的条件，那么该员工就可以获得 106 美元的残疾津贴。

如果员工符合残疾津贴的条件，那么该员工就可以获得 106 美元的残疾津贴。

10

如果员工符合残疾津贴的条件，那么该员工就可以获得 106 美元的残疾津贴。

```
function disabilityAmount(anEmployee) {
    if (anEmployee.seniority < 2) return 0;
    if (anEmployee.monthsDisabled > 12) return 0;
    if (anEmployee.isPartTime) return 0;
    // compute the disability amount
}
```



```
function getPayAmount() {
  let result;
  if (isDead) result = deadAmount();
  else {
    if (isSeparated) result = separatedAmount();
    else {
      if (isRetired) result = retiredAmount();
      else result = normalPayAmount();
    }
  }
  return result;
}

function getPayAmount() {
  if (isDead) return deadAmount();
  if (isSeparated) return separatedAmount();
  if (isRetired) return retiredAmount();
  return normalPayAmount();
}
```

## 12

guard clauses

if...else... guard clauses

if-then-else

guard clauses

## 13

guard clauses

guard clauses

guard clauses

guard clauses 263

## 14

employee









```
switch (bird.type) {
  case 'EuropeanSwallow':
    return "average";
  case 'AfricanSwallow':
    return (bird.numberOfCoconuts > 2) ? "tired" : "average";
  case 'NorwegianBlueParrot':
    return (bird.voltage > 100) ? "scorched" : "beautiful";
  default:
    return "unknown";
}

class EuropeanSwallow {
  get plumage() {
    return "average";
  }
}
class AfricanSwallow {
  get plumage() {
    return (this.numberOfCoconuts > 2) ? "tired" : "average";
  }
}
class NorwegianBlueParrot {
  get plumage() {
    return (this.voltage > 100) ? "scorched" : "beautiful";
  }
}
```

## 11

switch/case 语句的语法如下：

switch 语句的语法如下：

switch 语句的语法如下：

switch 语句的语法如下：

## 11

switch 语句的语法如下：

switch 语句的语法如下：

switch 语句的语法如下：

switch 语句的语法如下：

switch 语句的语法如下：

switch 语句的语法如下：

switch 语句的语法如下：



□ □

plumage switch



□ □ □ □ □ □ □ □ □ □ □



“‘’”——  
“”  
144  
“”

◀ ▶

```
class ExperiencedChinaRating extends Rating {}
```

```
function createRating(voyage, history) {
  if (voyage.zone === "china" &&& history.some(v => "china" === v.zone))
    return new ExperiencedChinaRating(voyage, history);
  else return new Rating(voyage, history);
}
```





□□□□□□□□106□□□□□□□□□□□□□□

## class Rating...

[illegible]

```
class ExperiencedChinaRating...
```

`"And"`106`"history`  
`length`

## class Rating...

```
get voyageAndHistoryLengthFactor() {
  let result = 0;
  result += this.historyLengthFactor;
  if (this.voyage.length > 14) result -= 1;
  return result;
}

get historyLengthFactor() {
  return (this.history.length > 8) ? 1 : 0;
}
```

□□□□□□□□□□

```
class ExperiencedChinaRating...
```

```
get voyageAndHistoryLengthFactor() {
  let result = 0;
  result += 3;
  result += this.historyLengthFactor;
  if (this.voyage.length > 12) result += 1;
  if (this.voyage.length > 18) result -= 1;
  return result;
}

get historyLengthFactor() {
  return (this.history.length > 10) ? 1 : 0;
}
```

217

## class Rating...

```

get voyageProfitFactor() {
  let result = 2;
  if (this.voyage.zone === "china") result += 1;
  if (this.voyage.zone === "east-indies") result += 1;
  result += this.historyLengthFactor;
  result += this.voyageAndHistoryLengthFactor;
  return result;
}

get voyageAndHistoryLengthFactor() {
  let result = 0;
  result += this.historyLengthFactor;
  if (this.voyage.length > 14) result -= 1;
  return result;
}

```

## class ExperiencedChinaRating...

```

get voyageAndHistoryLengthFactor() {
  let result = 0;
  result += 3;
  result += this.historyLengthFactor;
  if (this.voyage.length > 12) result += 1;
  if (this.voyage.length > 18) result -= 1;
  return result;
}

```

□□□□□□□□124□□□□□□□□□□□□

## class Rating...

```

get voyageProfitFactor() {
  let result = 2;
  if (this.voyage.zone === "china") result += 1;
  if (this.voyage.zone === "east-indies") result += 1;
  result += this.historyLengthFactor;
  result += this.voyageLengthFactor;
  return result;
}

get voyageLengthFactor() {
  return (this.voyage.length > 14) ? - 1: 0;
}

```

□□□□□□□□□□ voyageLengthFactor □□□

## class ExperiencedChinaRating...

```

get voyageLengthFactor() {
  let result = 0;
  result += 3;
  if (this.voyage.length > 12) result += 1;
  if (this.voyage.length > 18) result -= 1;
  return result;
}

```

□□□□□□□□“□□□”□voyage length□□□□□□ 3 □□□□□□□□□□□□□□□□□□ 3 □□□□□□□□□□□□



## 10.5 Introduce Special Case

□ □



```
const weeksDelinquent =
  aCustomer === "unknown"
    ? 0
    : aCustomer.paymentHistory.weeksDelinquentInLastYear;
```

Site “occupant” “Special Case Object” Customer “occupant”

## class Customer...

```
get isUnknown() {return false;}
```

“UnknownCustomer”

```
class UnknownCustomer {
  get isUnknown() {
    return true;
  }
}
```

UnknownCustomer Customer JavaScript

“unknown” “unknown” isUnknown Customer UnknownCustomer “unknown” isUnknown

“UnknownCustomer” 106

```
function isUnknown(arg) {
  if (!(arg instanceof Customer || arg === "unknown"))
    throw new Error(`investigate bad value: <${arg}>`);
  return arg === "unknown";
}
```

“UnknownCustomer” “UnknownCustomer”

## 1...

```
let customerName;
if (isUnknown(aCustomer)) customerName = "occupant";
else customerName = aCustomer.name;
```

“UnknownCustomer”



□□□ 2...

```
const plan = isUnknown(aCustomer)
  ? registry.billingPlans.basic
  : aCustomer.billingPlan;
```

**3...**

```
if (!isUnknown(aCustomer)) aCustomer.billingPlan = newPlan;
```

□□□ 4...

```
const weeksDelinquent = isUnknown(aCustomer)
  ? 0
  : aCustomer.paymentHistory.weeksDelinquentInLastYear;
```

XXXXXXXXXX isUnknown XXXXXXXXXX Site XXXXXXXXXX  
UnknownCustomer XXX

## class Site...

```
get_customer() {
    return (this._customer === "unknown") ? new UnknownCustomer() : this._customer;
}
```

```
isUnknown ""unknown""
```

□□□ **1...**

```
function isUnknown(arg) {
  if (!(arg instanceof Customer || arg instanceof UnknownCustomer))
    throw new Error(`investigate bad value: <${arg}>`);
  return arg.isUnknown;
}
```

□ □ □ □ □ □ □ □ □ □ □ □ □ □

[illegible]

1...

```
let customerName;  
if (isUnknown(aCustomer)) customerName = "occupant";  
else customerName = aCustomer.name;
```

UnknownCustomer 000000000000

## class UnknownCustomer...

```
get name() {return "occupant";}
```

~~~~~

### 1...

```
const customerName = aCustomer.name;
```

~~~~~123~~~~~ customerName ~~~~~

~~~~~“~~~~~”~~~~~ billingPlan ~~~~~

### 2...

```
const plan = isUnknown(aCustomer)
  ? registry.billingPlans.basic
  : aCustomer.billingPlan;
```

### 3...

```
if (!isUnknown(aCustomer)) aCustomer.billingPlan = newPlan;
```

~~~~~ name ~~~~~——~~~~~

UnknownCustomer ~~~~~

~~~~~

## class UnknownCustomer...

```
get billingPlan() {return registry.billingPlans.basic;}
set billingPlan(arg) { /* ignore */ }
```

### ~~~~~...

```
const plan = aCustomer.billingPlan;
```

### ~~~~~...

```
aCustomer.billingPlan = newPlan;
```

~~~~~

~~~~~

### ~~~~~...

```
const weeksDelinquent = isUnknown(aCustomer)
  ? 0
  : aCustomer.paymentHistory.weeksDelinquentInLastYear;
```

NullPaymentHistory

## class UnknownCustomer...

```
get paymentHistory() {return new NullPaymentHistory();}
```

## class NullPaymentHistory...

```
get weeksDelinquentInLastYear() {return 0;}
```

...

```
const weeksDelinquent = aCustomer.paymentHistory.weeksDelinquentInLastYear;
```

23 "occupant"

...

```
const name = !isUnknown(aCustomer) ? aCustomer.name : "unknown occupant";
```

aCustomer isUnknown  
isUnknown 115

...

```
const name = aCustomer.isUnknown ? "unknown occupant" : aCustomer.name;
```

isUnknown 237

Customer literal  
object

Customer

## class Site...

```
get customer() {return this._customer;}
```

11

## class Customer...

```
get name() { ... }
get billingPlan() { ... }
set billingPlan(arg) { ... }
get paymentHistory() { ... }
```

□□□ **1...**

```
const aCustomer = site.customer;
// ... lots of intervening code ...
let customerName;
if (aCustomer === "unknown") customerName = "occupant";
else customerName = aCustomer.name;
```

□□□ 2...

```
const plan =
  aCustomer === "unknown" ? registry.billingPlans.basic : aCustomer.billingPlan
```

□□□ **3...**

```
const weeksDelinquent =
  aCustomer === "unknown"
    ? 0
    : aCustomer.paymentHistory.weeksDelinquentInLastYear;
```

```

XXXXXXXXXXXXXXXX Customer XXX isUnknown XXXXXXXXXXXXXXXXXXXXXXXXXXXX
XXXXXXXXXXXXXXXX

```

## class Customer...

```
get isUnknown() {return false;}
```

□□□□□...

```
function createUnknownCustomer() {
    return {
        isUnknown: true,
    };
}
```

106

```
function isUnknown(arg) {  
    return arg === "unknown";  
}
```

□□□ **1...**

11

```
let customerName;  
if (isUnknown(aCustomer)) customerName = "occupant";  
else customerName = aCustomer.name;
```

□□□ 2...

```
const plan = isUnknown(aCustomer)
  ? registry.billingPlans.basic
  : aCustomer.billingPlan;
```

□□□ **3...**

```
const weeksDelinquent = isUnknown(aCustomer)
  ? 0
  : aCustomer.paymentHistory.weeksDelinquentInLastYear;
```

Site isUnknown

## class Site...

```
get_customer() {  
    return (this._customer === "unknown") ? createUnknownCustomer() : this._customer;  
}
```

□□□□□...

```
function isUnknown(arg) {
    return arg.isUnknown;
}
```

```
function createUnknownCustomer() {  
  return {  
    isUnknown: true,  
    name: "occupant",  
  };  
}
```

□□□ **1...**

```
const customerName = aCustomer.name;
```

□□□□“□□□□”□

11

```
function createUnknownCustomer() {
  return {
    isUnknown: true,
    name: "occupant",
    billingPlan: registry.billingPlans.basic,
  };
}
```

□□□ **2...**

```
const plan = aCustomer.billingPlan;
```

```
function createUnknownCustomer() {
  return {
    isUnknown: true,
    name: "occupant",
    billingPlan: registry.billingPlans.basic,
    paymentHistory: {
      weeksDelinquentInLastYear: 0,
    },
  };
}
```

**3...**

```
const weeksDelinquent = aCustomer.paymentHistory.weeksDelinquentInLastYear;
```

```

Object.freeze

```

□ □ □ □ □ □ □

```
{
  name: "Acme Boston",
  location: "Malden MA",
  // more site details
  customer: {
    name: "Acme Industries",
    billingPlan: "plan-451",
    paymentHistory: {
      weeksDelinquentInLastYear: 7
    }
  },
  // more
}
}
```

```
customer ""unknown""
```



11

```
const rawSite = acquireSiteData();
const site = enrichSite(rawSite);
const aCustomer = site.customer;
// ... lots of intervening code ...
let customerName;
if (isUnknown(aCustomer)) customerName = "occupant";
else customerName = aCustomer.name;
```

□□□ **2...**

```
const plan = isUnknown(aCustomer)
  ? registry.billingPlans.basic
  : aCustomer.billingPlan;
```

□□□ 3...

```
const weeksDelinquent = isUnknown(aCustomer)
  ? 0
  : aCustomer.paymentHistory.weeksDelinquentInLastYear;
```

Site customer isUnknown

```
function enrichSite(aSite) {
  const result = _.cloneDeep(aSite);
  const unknownCustomer = {
    isUnknown: true,
  };

  if (isUnknown(result.customer)) result.customer = unknownCustomer;
  else result.customer.isUnknown = false;
  return result;
}
```

Site 1

```
function isUnknown(aCustomer) {
    if (aCustomer === "unknown") return true;
    else return aCustomer.isUnknown;
}
```

149 “ ”

```
function enrichSite(aSite) {
  const result = _.cloneDeep(aSite);
  const unknownCustomer = {
    isUnknown: true,
    name: "occupant",
  };

  if (isUnknown(result.customer)) result.customer = unknownCustomer;
  else result.customer.isUnknown = false;
  return result;
}
```

□□□ **1...**



```
const rawSite = acquireSiteData();
const site = enrichSite(rawSite);
const aCustomer = site.customer;
// ... lots of intervening code ...
const customerName = aCustomer.name;
```

□□□□□□“□□□□□□□□□□”□□□□□□□□

```
function enrichSite(aSite) {
  const result = _.cloneDeep(aSite);
  const unknownCustomer = {
    isUnknown: true,
    name: "occupant",
    billingPlan: registry.billingPlans.basic,
  };

  if (isUnknown(result.customer)) result.customer = unknownCustomer;
  else result.customer.isUnknown = false;
  return result;
}
```

□□□ 2...

```
const plan = aCustomer.billingPlan;
```

□□□□□□□□□□□□□□□□□□□□□□□□

```
function enrichSite(aSite) {
  const result = _.cloneDeep(aSite);
  const unknownCustomer = {
    isUnknown: true,
    name: "occupant",
    billingPlan: registry.billingPlans.basic,
    paymentHistory: {
      weeksDelinquentInLastYear: 0,
    },
  };

  if (isUnknown(result.customer)) result.customer = unknownCustomer;
  else result.customer.isUnknown = false;
  return result;
}
```

□□□ 3...

```
const weeksDelinquent = aCustomer.paymentHistory.weeksDelinquentInLastYear;
```

## 10.6 □□□□□Introduce Assertion□

```
if (this.discountRate)
  base = base - (this.discountRate * base);

assert(this.discountRate >= 0);
if (this.discountRate)
  base = base - (this.discountRate * base);
```

class Customer...

applyDiscount(aNumber) {  
return (this.discountRate  
? aNumber - (this.discountRate \* aNumber)  
: aNumber;  
}

class Customer...  
applyDiscount(aNumber) {  
return (this.discountRate  
? aNumber - (this.discountRate \* aNumber)  
: aNumber;  
}

class Customer...  
applyDiscount(aNumber) {  
return (this.discountRate  
? aNumber - (this.discountRate \* aNumber)  
: aNumber;  
}

class Customer...  
applyDiscount(aNumber) {  
return (this.discountRate  
? aNumber - (this.discountRate \* aNumber)  
: aNumber;  
}

class Customer...

applyDiscount(aNumber) {  
return (this.discountRate  
? aNumber - (this.discountRate \* aNumber)  
: aNumber;  
}

class Customer...  
applyDiscount(aNumber) {  
return (this.discountRate  
? aNumber - (this.discountRate \* aNumber)  
: aNumber;  
}

class Customer...

applyDiscount(aNumber) {  
return (this.discountRate  
? aNumber - (this.discountRate \* aNumber)  
: aNumber;  
}

class Customer...

```
applyDiscount(aNumber) {  
  return (this.discountRate  
    ? aNumber - (this.discountRate * aNumber)  
    : aNumber;  
}
```

class Customer...  
applyDiscount(aNumber) {  
return (this.discountRate  
? aNumber - (this.discountRate \* aNumber)  
: aNumber;  
}

class Customer...

```
applyDiscount(aNumber) {  
  if (!this.discountRate) return aNumber;  
  else return aNumber - (this.discountRate * aNumber);  
}
```

class Customer...

class Customer...

11

```

applyDiscount(aNumber) {
  if (!this.discountRate) return aNumber;
  else {
    assert(this.discountRate >= 0);
    return aNumber - (this.discountRate * aNumber);
  }
}

```

```

    applyDiscount
    
```

## class Customer...

```
set discountRate(aNumber) {
    assert(null == aNumber || aNumber >= 0);
    this._discountRate = aNumber;
}
```

[illegible][illegible][illegible]

## 11 11 API

API API API  
API API API

API 306 310 314

319  
 324  
 327

331

334

337 106 344

## 11.1 Separate Query from Modifier

```
function getTotalOutstandingAndSendBill() {
  const result = customer.invoices.reduce((total, each) => each.amount + total, 0);
  sendBill();
  return result;
}

function totalOutstanding() {
  return customer.invoices.reduce((total, each) => each.amount + total, 0);
}

function sendBill() {
  emailGateway.send(formatBill(customer));
}
```

◀ ▶

11

[illegible]

“ ” “ ”  
——Command-Query Separation[mf-cqs]

□□□□□“□□□□□□□□□”□□□□□□□□□□□□□□□□□□□□□□□□

1. 如果 people 数组中有人是 "Don"，那么返回 "Don"。  
 2. 如果 people 数组中有人是 "John"，那么返回 "John"。  
 3. 如果 people 数组中没有人是 "Don" 或 "John"，那么返回 ""。

11

1. 如果 people 数组中有人是 "Don"，那么返回 "Don"。

2. 如果 people 数组中有人是 "John"，那么返回 "John"。

3. 如果 people 数组中没有人是 "Don" 或 "John"，那么返回 ""。

11

1. 如果 people 数组中有人是 "Don"，那么返回 "Don"。  
 2. 如果 people 数组中有人是 "John"，那么返回 "John"。  
 3. 如果 people 数组中没有人是 "Don" 或 "John"，那么返回 ""。

11

11

1. 如果 people 数组中有人是 "Don"，那么返回 "Don"。

11

1. 如果 people 数组中有人是 "Don"，那么返回 "Don"。  
 2. 如果 people 数组中有人是 "John"，那么返回 "John"。  
 3. 如果 people 数组中没有人是 "Don" 或 "John"，那么返回 ""。

```

function alertForMiscreant(people) {
  for (const p of people) {
    if (p === "Don") {
      setOffAlarms();
      return "Don";
    }
    if (p === "John") {
      setOffAlarms();
      return "John";
    }
  }
  return "";
}
    
```

1. 如果 people 数组中有人是 "Don"，那么返回 "Don"。

```

function findMiscreant(people) {
  for (const p of people) {
    if (p === "Don") {
      setOffAlarms();
      return "Don";
    }
    if (p === "John") {
      setOffAlarms();
      return "John";
    }
  }
  return "";
}
    
```

1. 如果 people 数组中有人是 "Don"，那么返回 "Don"。

11

```
function findMiscreant(people) {
  for (const p of people) {
    if (p === "Don") {
      setOffAlarms();
      return "Don";
    }
    if (p === "John") {
      setOffAlarms();
      return "John";
    }
  }
  return "";
}
```

```
const found = alertForMiscreant(people);
```

11

```
const found = findMiscreant(people);
alertForMiscreant(people);
```

□□□□□□□□□□□□□□□□

```
function alertForMiscreant(people) {
  for (const p of people) {
    if (p === "Don") {
      setOffAlarms();
      return;
    }
    if (p === "John") {
      setOffAlarms();
      return;
    }
  }
  return;
}
```

195

```
function alertForMiscreant(people) {
  if (findMiscreant(people) !== "") setOffAlarms();
}
```

## 11.2 Parameterize Function

## Parameterize Method

11

```
function tenPercentRaise(aPerson) {
  aPerson.salary = aPerson.salary.multiply(1.1);
}

function fivePercentRaise(aPerson) {
  aPerson.salary = aPerson.salary.multiply(1.05);
}

function raise(aPerson, factor) {
  aPerson.salary = aPerson.salary.multiply(1 + factor);
}
```

11

[illegible]

11

□□□□□□□□□□

124

□ □ □ □ □ □ □ □ □ □ □ □ □ □ □ □ □ □ □ □ □ □ □ □

111

□ □ □ □ □ □ □ □ □ □ □ □ □ □ □ □ □ □ □ □ □ □ □ □

[illegible]

11

□□□□□□□□□□□□

```
function tenPercentRaise(aPerson) {
  aPerson.salary = aPerson.salary.multiply(1.1);
}
function fivePercentRaise(aPerson) {
  aPerson.salary = aPerson.salary.multiply(1.05);
}
```

```
function raise(aPerson, factor) {
  aPerson.salary = aPerson.salary.multiply(1 + factor);
}
```

239

```

uint8_t *data = (uint8_t *) malloc(1024);
uint8_t *middleBand = (uint8_t *) malloc(1024);

middleBand[0] = 100;
middleBand[1] = 200;
middleBand[2] = 124;
middleBand[3] = 0;

```

□ □ □ □ □ □ □ □ □ □ □ □ □ □ □ □ □ □ □ □

□□□□□□

bottomBand





```
function bookConcert(aCustomer, isPremium) {
  if (isPremium) {
    // logic for premium booking
  } else {
    // logic for regular booking
  }
}
```

premium concert

```
bookConcert(aCustomer, true);
```

```
bookConcert(aCustomer, CustomerType.PREMIUM);
```

```
bookConcert(aCustomer, "premium");
```

API true

```
premiumBookConcert(aCustomer);
```

260

shipment delivery date

```
aShipment.deliveryDate = deliveryDate(anOrder, true);
```



deliveryDate  
deliveryDate

```
const isRush = determineIfRush(anOrder);
aShipment.deliveryDate = deliveryDate(anOrder, isRush);
```

260

314 deliveryDate

deliveryDate

```
function deliveryDate(anOrder, isRush) {
  let result;
  let deliveryTime;
  if (anOrder.deliveryState === "MA" || anOrder.deliveryState === "CT")
    deliveryTime = isRush ? 1 : 2;
  else if (anOrder.deliveryState === "NY" || anOrder.deliveryState === "NH") {
    deliveryTime = 2;
    if (anOrder.deliveryState === "NH" && !isRush)
      deliveryTime = 3;
  }
  else if (isRush)
    deliveryTime = 3;
  else if (anOrder.deliveryState === "ME")
    deliveryTime = 3;
  else
    deliveryTime = 4;
  result = anOrder.placedOn.plusDays(2 + deliveryTime);
  if (isRush) result = result.minusDays(1);
  return result;
}
```

isRush deliveryDate

```
function rushDeliveryDate(anOrder) {
  return deliveryDate(anOrder, true);
}
function regularDeliveryDate(anOrder) {
  return deliveryDate(anOrder, false);
}
```

deliveryDate

isRush deliveryDateHelperOnly

## 11.4 Preserve Whole Object















## 11.6 Replace Query with Parameter

□□□□□□□□□□324□

```
targetTemperature(aPlan)

function targetTemperature(aPlan) {
  currentTemperature = thermostat.currentTemperature;
  // rest of function...

  targetTemperature(aPlan, thermostat.currentTemperature)
}

function targetTemperature(aPlan, currentTemperature) {
  // rest of function...
```

11

[illegible]

□□□□□□□□□□□□□□□□□□——□□□□□□□□□□□□□□□□□□□□  
□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□  
□□□□□□□□□□□□□□□□□□“□□□□□□”□□□□□□□□□□□□□□□□□□

transparency

I/O

[illegible]

11

119

106

□ □ □ □ □ □ □ □ □ □ □ □ □ □ □ □ □ □ □ □ □ □ □ □

123

□□□□□□□□□□115□□

□ □ □ □ □ □ □ □ □ □ □ □ □ □ □ □

11

```

#####thermostat#####
###heating plan###

```







Person 对象的 id 属性

```
const martin = new Person("1234");
martin.name = "martin";
martin.id = "1234";
```

Person 对象的 id 属性

Person 对象的 id 属性 115

## class Person...

```
constructor(id) {
  this._id = id;
}
get name() {return this._name;}
set name(arg) {this._name = arg;}
get id() {return this._id;}
set id(arg) {this._id = arg;}
```

## 11.8 Replace Constructor with Factory Function

Replace Constructor with Factory Method

```
leadEngineer = new Employee(document.leadEngineer, "E");
leadEngineer = createEngineer(document.leadEngineer);
```

11

Person 对象的 id 属性 115

Person 对象的 id 属性 115

11

Person 对象的 id 属性 115

Person 对象的 id 属性 115

Person 对象的 id 属性 115

Person 对象的 id 属性 115

11

Person 对象的 id 属性 115

## class Employee...





```
const leadEngineer = createEngineer(document.leadEngineer);
```

~~~~~

```
function createEngineer(name) {
  return new Employee(name, "E");
}
```

## 11.9 Replace Function with Command

Replace Method with Method Object

344

```
function score(candidate, medicalExam, scoringGuide) {
  let result = 0;
  let healthLevel = 0;
  // long body code
}

class Scorer {
  constructor(candidate, medicalExam, scoringGuide) {
    this._candidate = candidate;
    this._medicalExam = medicalExam;
    this._scoringGuide = scoringGuide;
  }

  execute() {
    this._result = 0;
    this._healthLevel = 0;
    // long body code
  }
}
```

11

method command object command object

command

95%

command pattern

11

11

198

11

```
function score(candidate, medicalExam, scoringGuide) {
  let result = 0;
  let healthLevel = 0;
  let highMedicalRiskFlag = false;

  if (medicalExam.isSmoker) {
    healthLevel += 10;
    highMedicalRiskFlag = true;
  }

  let certificationGrade = "regular";
  if (scoringGuide.stateWithLowCertification(candidate.originState)) {
    certificationGrade = "low";
    result -= 5;
  } // lots more code like this

  result -= Math.max(healthLevel - 5, 0);
  return result;
}
```

198

[illegible]

◀ ▶

## class Scorer...

□ □ □ □ □ □ □ □ □



```
function score(candidate, medicalExam, scoringGuide) {
  return new Scorer(candidate, medicalExam, scoringGuide).execute();
}
```

## class Scorer...

```

constructor(candidate, medicalExam, scoringGuide){
  this._candidate = candidate;
  this._medicalExam = medicalExam;
  this._scoringGuide = scoringGuide;
}

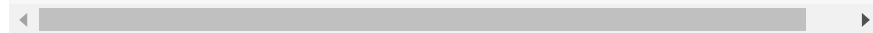
execute () {
  let result = 0;
  let healthLevel = 0;
  let highMedicalRiskFlag = false;

  if (this._medicalExam.isSmoker) {
    healthLevel += 10;
    highMedicalRiskFlag = true;
  }

  let certificationGrade = "regular";
  if (this._scoringGuide.stateWithLowCertification(this._candidate.originState))
    certificationGrade = "low";
    result -= 5;
  }

  // lots more code like this
  result -= Math.max(healthLevel - 5, 0);
  return result;
}

```

[illegible]

## class Scorer...

```

constructor(candidate, medicalExam, scoringGuide){
  this._candidate = candidate;
  this._medicalExam = medicalExam;
  this._scoringGuide = scoringGuide;
}

execute () {
  this._result = 0;
  let healthLevel = 0;
  let highMedicalRiskFlag = false;

  if (this._medicalExam.isSmoker) {
    healthLevel += 10;
    highMedicalRiskFlag = true;
  }

  let certificationGrade = "regular";
  if (this._scoringGuide.stateWithLowCertification(this._candidate.originState))
    certificationGrade = "low";
  this._result -= 5;
}

// lots more code like this
this._result -= Math.max(healthLevel - 5, 0);
return this._result;
}

```





□□□□□□□□□□□□□□□□□□□□“□□□□□□□□”□□□□□□□□□□□□□□□□□□□□□□□□  
□□□□□□□□□□□□

## class Scorer...

```

constructor(candidate, medicalExam, scoringGuide){
    this._candidate = candidate;
    this._medicalExam = medicalExam;
    this._scoringGuide = scoringGuide;
}

execute () {
    this._result = 0;
    this._healthLevel = 0;
    this._highMedicalRiskFlag = false;

    if (this._medicalExam.isSmoker) {
        this._healthLevel += 10;
        this._highMedicalRiskFlag = true;
    }

    this._certificationGrade = "regular";
    if (this._scoringGuide.stateWithLowCertification(this._candidate.originState))
        this._certificationGrade = "low";
    this._result -= 5;
}

// lots more code like this
this._result -= Math.max(this._healthLevel - 5, 0);
return this._result;
}

```

106

## class Scorer...

```
execute () {
  this._result = 0;
  this._healthLevel = 0;
  this._highMedicalRiskFlag = false;

  this.scoreSmoking();
  this._certificationGrade = "regular";
  if (this._scoringGuide.stateWithLowCertification(this._candidate.originState)) {
    this._certificationGrade = "low";
    this._result -= 5;
  }
  // lots more code like this
  this._result -= Math.max(this._healthLevel - 5, 0);
  return this._result;
}

scoreSmoking() {
  if (this._medicalExam.isSmoker) {
    this._healthLevel += 10;
    this._highMedicalRiskFlag = true;
  }
}
```

JavaScript

## 11.10 Replace Command with Function

337

```
class ChargeCalculator {
  constructor(customer, usage) {
    this._customer = customer;
    this._usage = usage;
  }
  execute() {
    return this._customer.rate * this._usage;
  }
}

function charge(customer, usage) {
  return customer.rate * usage;
}
```

11

106 “ ”

11

106 “ ”

115

119 115

124

119 115

124

119 115

11

237

11

237

```
class ChargeCalculator {
  constructor(customer, usage, provider) {
    this._customer = customer;
    this._usage = usage;
    this._provider = provider;
  }
  get baseCharge() {
    return this._customer.baseRate * this._usage;
  }
  get charge() {
    return this.baseCharge + this._provider.connectionCharge;
  }
}
```

□□□□□□□□

□□□...

```
monthCharge = new ChargeCalculator(customer, usage, provider).charge;
```

□□□□□□□□□□□□□□□□□□

□□□□□□□□106□□□□□□□□□□□□□□□□□□□□□□

□□□...

```
monthCharge = charge(customer, usage, provider);
```

□□□□□...

```
function charge(customer, usage, provider) {
  return new ChargeCalculator(customer, usage, provider).charge;
}
```

□□□□□□□□□□□□□□□□□□ baseCharge □□□□□□□□□□□□□□□□□□□□□□119□  
□□□□□□□□

**class ChargeCalculator...**

```
get baseCharge() {
  return this._customer.baseRate * this._usage;
}
get charge() {
  const baseCharge = this.baseCharge;
  return baseCharge + this._provider.connectionCharge;
}
```

□□□□□□□□□□□□□□115□□

**class ChargeCalculator...**

```
get charge() {
  const baseCharge = this._customer.baseRate * this._usage;
  return baseCharge + this._provider.connectionCharge;
}
```

124

charge

```
class ChargeCalculator...
```

```
constructor (customer, usage, provider){
  this._customer = customer;
  this._usage = usage;
  this._provider = provider;
}

charge(customer, usage, provider) {
  const baseCharge = this._customer.baseRate * this._usage;
  return baseCharge + this._provider.connectionCharge;
}
```

□ □ □ □ □ ...

```
function charge(customer, usage, provider) {
    return new ChargeCalculator(customer, usage, provider).charge(
        customer,
        usage,
        provider
    );
}
```

charge

```
class ChargeCalculator...
```

```
constructor (customer, usage, provider){
  this._customer = customer;
  this._usage = usage;
  this._provider = provider;
}

charge(customer, usage, provider) {
  const baseCharge = customer.baseRate * this._usage;
  return baseCharge + this._provider.connectionCharge;
}
```

charge

```
class ChargeCalculator...
```



11

```
charge(customer, usage, provider) {
  const baseCharge = customer.baseRate * usage;
  return baseCharge + provider.connectionCharge;
}
```

charge 115

□□□□□...

```
function charge(customer, usage, provider) {
  const baseCharge = customer.baseRate * usage;
  return baseCharge + provider.connectionCharge;
}
```

□□□□□□□□□□□□□□□□□□□□237□□□□□□□□□□

**12**

## 12.1 Pull Up Method

□□□□□□□□359□

```
class Employee {...}

class Salesman extends Employee {
    get name() {...}
}

class Engineer extends Employee {
    get name() {...}
}

class Employee {
    get name() {...}
}

class Salesman extends Employee {...}
class Engineer extends Employee {...}
```

11

310

353

Form Template Method  
[mf-ft]

11



11

```

// JavaScript 関数定義
// Employee 関数 annualCost 関数 Department 関数
annualCost 関数

```

## class Party...

```
get monthlyCost() {
    throw new SubclassResponsibilityError();
}
```

□□□□□□□□□□“□□□□□□□□□□”□□□□ Smalltalk □□□□□□□□

## 12.2 Pull Up Field

□□□□□□□□361□

```
class Employee {...} // Java

class Salesman extends Employee {
    private String name;
}

class Engineer extends Employee {
    private String name;
}

class Employee {
    protected String name;
}

class Salesman extends Employee {...}
class Engineer extends Employee {...}
```

11

11

137

protected

## 12.3 Pull Up Constructor Body

```
class Party {...}

class Employee extends Party {
  constructor(name, id, monthlyCost) {
    super();
    this._id = id;
    this._name = name;
    this._monthlyCost = monthlyCost;
  }
}

class Party {
  constructor(name){
    this._name = name;
  }
}

class Employee extends Party {
  constructor(name, id, monthlyCost) {
    super(name);
    this._id = id;
    this._monthlyCost = monthlyCost;
  }
}
```

106

350

334

223



□□□□□□□□□□

□□□□□□□□

```
constructor (name) {...}

get isPrivileged() {...}

assignCar() {...}
```

```
constructor(name, grade) {  
  super(name);  
  this._grade = grade;  
  if (this.isPrivileged) this.assignCar(); // every subclass does this  
}  
  
get isPrivileged() {  
  return this._grade > 4;  
}
```

106

```
constructor(name, grade) {
  super(name);
  this._grade = grade;
  this.finishConstruction();
}

finishConstruction() {
  if (this.isPrivileged) this.assignCar();
}
```

## class Employee...













## class Employee...

```

constructor(name, type){
  this.validateType(type);
  this._name = name;
  this._type = type;
}
validateType(arg) {
  if (!["engineer", "manager", "salesman"].includes(arg))
    throw new Error(`Employee cannot be of type ${arg}`);
}
get type() {return this._type;}
set type(arg) {this._type = arg;}

get capitalizedType() {
  return this._type.charAt(0).toUpperCase() + this._type.substr(1).toLowerCase();
}
toString() {
  return `${this._name} (${this.capitalizedType})`;
}

```

□□□ toString □□□□□□□□□□□□□□□□

□□□□□□□□□□□□□□□□174□□□□□□□□

```

class EmployeeType {
  constructor(aString) {
    this._value = aString;
  }
  toString() {
    return this._value;
  }
}

```

## class Employee...

```

constructor(name, type){
  this.validateType(type);
  this._name = name;
  this.type = type;
}
validateType(arg) {
  if (!["engineer", "manager", "salesman"].includes(arg))
    throw new Error(`Employee cannot be of type ${arg}`);
}
get typeString() {return this._type.toString();}
get type() {return this._type;}
set type(arg) {this._type = new EmployeeType(arg);}

get capitalizedType() {
  return this.typeString.charAt(0).toUpperCase()
    + this.typeString.substr(1).toLowerCase();
}
toString() {
  return `${this._name} (${this.capitalizedType})`;
}

```

□□□□□□□□□□□□362□□□□□□□□□□□□□□□□

## class Employee...

```

set type(arg) {this._type = Employee.createEmployeeType(arg);}

static createEmployeeType(aString) {
  switch(aString) {
    case "engineer": return new Engineer();
    case "manager":  return new Manager ();
    case "salesman": return new Salesman();
    default: throw new Error(`Employee cannot be of type ${aString}`);
  }
}

class EmployeeType {
}
class Engineer extends EmployeeType {
  toString() {return "engineer";}
}
class Manager extends EmployeeType {
  toString() {return "manager";}
}
class Salesman extends EmployeeType {
  toString() {return "salesman";}
}

```

EmployeeType 的 createEmployeeType 方法，根据传入的字符串，返回相应的 EmployeeType 子类实例。toString 方法，返回子类的名称。

## class Employee...

```

toString() {
  return `${this._name} (${this.type.capitalizedName})`;
}

```

## class EmployeeType...

```

get capitalizedName() {
  return this.toString().charAt(0).toUpperCase()
    + this.toString().substr(1).toLowerCase();
}

```

Employee 的 toString 方法，返回子类的名称。EmployeeType 的 capitalizedName 方法，返回子类的名称，首字母大写，其余字母小写。

## 12.7 Remove Subclass

Replace Subclass with Fields

362







```
function createPerson(aRecord) {
  let p;
  switch (aRecord.gender) {
    case 'M': p = new Male(aRecord.name); break;
    case 'F': p = new Female(aRecord.name); break;
    default: p = new Person(aRecord.name);
  }
  return p;
}

function loadFromInput(data) {
  const result = [];
  data.forEach(aRecord => {
    result.push(createPerson(aRecord));
  });
  return result;
}
```

123 createPerson

```
function createPerson(aRecord) {
  switch (aRecord.gender) {
    case "M":
      return new Male(aRecord.name);
    case "F":
      return new Female(aRecord.name);
    default:
      return new Person(aRecord.name);
  }
}
```

231 loadFromInput

```
function loadFromInput(data) {
  return data.map(aRecord => createPerson(aRecord));
}
```

instanceof — 106

...

```
const numberOfMales = people.filter(p => isMale(p)).length;

function isMale(aPerson) {return aPerson instanceof Male;}
```

198 Person

## class Person...

```
get isMale() {return this instanceof Male;}
```

...

```
const numberOfMales = people.filter(p => p.isMale).length;
```

[illegible]

## class Person...

```
function createPerson(aRecord) {  
  switch (aRecord.gender) {  
    case "M":  
      return new Person(aRecord.name, "M");  
    case "F":  
      return new Female(aRecord.name);  
    default:  
      return new Person(aRecord.name);  
  }  
}
```

```

Person
instanceof

```

☐ Male ☐ Female ☐

[illegible]

11

```
function createPerson(aRecord) {  
  switch (aRecord.gender) {  
    case "M":  
      return new Person(aRecord.name, "M");  
    case "F":  
      return new Person(aRecord.name, "F");  
    default:  
      return new Person(aRecord.name, "X");  
  }  
}
```

## class Person...

```
constructor(name, genderCode) {
  this._name = name;
  this._genderCode = genderCode || "X";
}
```

## 12.8 Extract Superclass

```
class Department {
  get totalAnnualCost() {...}
  get name() {...}
  get headCount() {...}
}

class Employee {
  get annualCost() {...}
  get name() {...}
  get id() {...}
}

class Party {
  get name() {...}
  get annualCost() {...}
}

class Department extends Party {
  get annualCost() {...}
  get headCount() {...}
}

class Employee extends Party {
  get annualCost() {...}
  get id() {...}
}
```

11

353

350

“ ”

182

399



```
class Party {}

class Employee extends Party {
  constructor(name, id, monthlyCost) {
    super();
    this._id = id;
    this._name = name;
    this._monthlyCost = monthlyCost;
  }
  // rest of class...
}

class Department extends Party {
  constructor(name, staff){
    super();
    this._name = name;
    this._staff = staff;
  }
  // rest of class...
```

□□□□□□□□□□□□□□□□ JavaScript □□□□□□□□□□□□□□□□□□□□□□ 353□□ name □□ □□□□□□

## class Party...

```
constructor(name){
  this._name = name;
}
```

## class Employee...

```
constructor(name, id, monthlyCost) {
  super(name);
  this._id = id;
  this._monthlyCost = monthlyCost;
}
```

## class Department...

```
constructor(name, staff){
  super(name);
  this._staff = staff;
}
```

□□□□□□□□□□□□□□□□□□□□ 350□□□□□□□□□□□□□□□□ name □□□

## class Party...

```
get name() {return this._name;}
```

## class Employee...

```
get name() {return this._name;}
```

## class Department...

11

```
get name() {return this._name;}
```

□□□□□□□□□□

## class Employee...

```
get annualCost() {  
    return this.monthlyCost * 12;  
}
```

```
class Department...
```

```
get totalAnnualCost() {  
    return this.totalMonthlyCost * 12;  
}
```

```

monthlyCost = totalMonthlyCost / 124

```

## class Department...

```
get totalAnnualCost() {
    return this.monthlyCost * 12;
}

get monthlyCost() { ... }
```

□□□□□□□□□□□□□□□□□□

## class Department...

```
get annualCost() {  
    return this.monthlyCost * 12;  
}
```

□□□□□□□□350□□□□□□□□□□

## class Party...

```
get annualCost() {
    return this.monthlyCost * 12;
}
```

## class Employee...

```
get annualCost() {  
    return this.monthlyCost * 12;  
}
```

11

```
class Department...
```

```
get annualCost() {
  return this.monthlyCost * 12;
}
```

## 12.9 Collapse Hierarchy

```
class Employee {...}
class Salesman extends Employee {...}

class Employeee {...}
```

11

[illegible]

11

□□□□□□□□□□□□□□

[illegible]

□□□□□□353□□□□□□361□□□□□□350□□□□□□359□□□□□□□□□□□□□□□□

□ □

□ □

111

## 12.10 Replace Subclass with Delegate

287

11

11

□ □



□□□□□□□□□□□□□□□□

198

237

375

100

111

□□□□□□□237□□□□□□

11

## class Booking...

```
constructor(show, date) {  
  this._show = show;  
  this._date = date;  
}
```

## class PremiumBooking extends Booking...

```
constructor(show, date, extras) {  
  super(show, date);  
  this._extras = extras;  
}
```

“ ” talkback

## class Booking...

```
get hasTalkback() {
  return this._show.hasOwnProperty('talkback') && !this.isPeakDay;
}
```

## class PremiumBooking...

```
get hasTalkback() {
  return this._show.hasOwnProperty('talkback');
}
```

□□□□□□□□□□□□□□□□PremiumBooking□□□□□□□□□□

## class Booking...

```
get basePrice() {
  let result = this._show.price;
  if (this.isPeakDay) result += Math.round(result * 0.15);
  return result;
}
```

## class PremiumBooking...

```
get basePrice() {
    return Math.round(super.basePrice + this._extras.premiumFee);
}
```

 PremiumBooking 

```
class PremiumBooking...
```

```
get hasDinner() {  
    return this._extras.hasOwnProperty('dinner') &&& !this.isPeakDay;  
}
```

Diagram illustrating a 16-bit bus system. The bus is divided into two sections: the first 8 bits are labeled "Data bus" and the last 8 bits are labeled "Address bus".

```

#####
#####“#####”#####
#####“#####”#####
aBooking.bePremium()##### HTTP #####
#####“#####”##### Booking #####
#####“#####”“#####”#####

```

10/10/2019

```
aBooking = new Booking(show, date);
```

□□□□□□□□□□

□ □ □ □ □ ...

□ □ □ □ □ ...

Booking 的 hasTalkback 属性，当 booking 的 show 属性为 true 且 booking 的 extras 属性为 true 且 booking 的 isPeakDay 属性为 false 时，返回 true，否则返回 false。

Booking 的 hasTalkback 属性，当 booking 的 show 属性为 true 且 booking 的 extras 属性为 true 且 booking 的 isPeakDay 属性为 false 时，返回 true，否则返回 false。

## class Booking...

```
get hasTalkback() {
    return this._show.hasOwnProperty('talkback') && !this.isPeakDay;
}
```

## class PremiumBooking...

```
get hasTalkback() {
    return this._show.hasOwnProperty('talkback');
}
```

198 年，Booking 的 hasTalkback 属性，当 booking 的 show 属性为 true 且 booking 的 extras 属性为 true 且 booking 的 isPeakDay 属性为 false 时，返回 true，否则返回 false。

## class PremiumBookingDelegate...

```
get hasTalkback() {
    return this._host._show.hasOwnProperty('talkback');
}
```

## class PremiumBooking...

```
get hasTalkback() {
    return this._premiumDelegate.hasTalkback;
}
```

Booking 的 hasTalkback 属性，当 booking 的 show 属性为 true 且 booking 的 extras 属性为 true 且 booking 的 isPeakDay 属性为 false 时，返回 true，否则返回 false。

## class PremiumBooking...

```
get hasTalkback() {
    return this._premiumDelegate.hasTalkback;
}
```

Booking 的 hasTalkback 属性，当 booking 的 show 属性为 true 且 booking 的 extras 属性为 true 且 booking 的 isPeakDay 属性为 false 时，返回 true，否则返回 false。

Booking 的 hasTalkback 属性，当 booking 的 show 属性为 true 且 booking 的 extras 属性为 true 且 booking 的 isPeakDay 属性为 false 时，返回 true，否则返回 false。

## class Booking...

```
basePrice
```

```
get basePrice() {
  let result = this._show.price;
  if (this.isPeakDay) result += Math.round(result * 0.15);
  return result;
}
```

```
get basePrice() {
    return Math.round(super.basePrice + this._extras.premiumFee);
}
```

```
get basePrice() {
  return (this._premiumDelegate
    ? this._premiumDelegate.basePrice
    : this._privateBasePrice;
}

get _privateBasePrice() {
  let result = this._show.price;
  if (this.isPeakDay) result += Math.round(result * 0.15);
  return result;
}
```

```
get basePrice() {
  return Math.round(this._host._privateBasePrice + this._extras.premiumFee);
}
```

## class Booking...

## class PremiumBookingDelegate...

□ □ □ □ □ □ □ □ □ □ □ □ □ □ □ □ □ □ □ □

□ □ □ □ □ □ □ □ □ □ □ □

## Booking

□ □ □ □ □

## class PremiumBooking extends Booking ...

[illegible]

0000000000bird0000000000000000“000”0wild000000“000”0captive000000  
 0000000000000000000000 Bird 00000000WildBird 0 CaptiveBird00000000000000  
 00000000“00”0“00”000000000000“0000”000000  
  
 0000000000000000000000000000——0000000000 EuropeanSwallow000000  
 00000000000000

```
class EuropeanSwallowDelegate {}
```



이제 이 클래스를 사용하여 데이터를 처리할 수 있습니다.

이제 이 클래스를 사용하여 데이터를 처리할 수 있습니다. data 객체를 사용하여 데이터를 처리할 수 있습니다. data.type을 사용하여 데이터를 처리할 수 있습니다.

## class Bird...

```
constructor(data) {
  this._name = data.name;
  this._plumage = data.plumage;
  this._speciesDelegate = this.selectSpeciesDelegate(data);
}

selectSpeciesDelegate(data) {
  switch(data.type) {
    case 'EuropeanSwallow':
      return new EuropeanSwallowDelegate();
    default: return null;
  }
}
```

이제 이 클래스를 사용하여 데이터를 처리할 수 있습니다. 198 EuropeanSwallow airSpeedVelocity

## class EuropeanSwallowDelegate...

```
get airSpeedVelocity() {return 35;}
```

## class EuropeanSwallow...

```
get airSpeedVelocity() {return this._speciesDelegate.airSpeedVelocity;}
```

이제 이 클래스를 사용하여 데이터를 처리할 수 있습니다. airSpeedVelocity

## class Bird...

```
get airSpeedVelocity() {
  return this._speciesDelegate ? this._speciesDelegate.airSpeedVelocity : null;
}
```

이제 이 클래스를 사용하여 데이터를 처리할 수 있습니다.

```
class EuropeanSwallow extends Bird {
  get airSpeedVelocity() {
    return this._speciesDelegate.airSpeedVelocity;
  }
}
```

이제 이 클래스를 사용하여 데이터를 처리할 수 있습니다.

```
function createBird(data) {
  switch (data.type) {
    case "EuropeanSwallow":
      return new EuropeanSwallow(data);
    case "AfricanSwallow":
      return new AfricanSwallow(data);
    case "NorwegianBlueParrot":
      return new NorwegianBlueParrot(data);
    default:
      return new Bird(data);
  }
}
```

□□□□ AfricanSwallow□□□□□□□□□□□□□□□□□□□□□□□□ data □□□

## class AfricanSwallowDelegate...

```
constructor(data) {
  this._numberOfCoconuts = data.numberOfCoconuts;
}
```

class Bird...

```
selectSpeciesDelegate(data) {
  switch(data.type) {
    case 'EuropeanSwallow':
      return new EuropeanSwallowDelegate();
    case 'AfricanSwallow':
      return new AfricanSwallowDelegate(data);
    default: return null;
  }
}
```

□□□□□□□□198□□ airSpeedVelocity □□□□□□□□

## class AfricanSwallowDelegate...

```
get airSpeedVelocity() {
  return 40 - 2 * this._numberOfCoconuts;
}
```

## class AfricanSwallow...

```
get airSpeedVelocity() {
  return this._speciesDelegate.airSpeedVelocity;
}
```

□□□ AfricanSwallow □□□

11

```
class AfricanSwallow extends Bird {  
    // all of the body ...  
}  
  
function createBird(data) {  
    switch (data.type) {  
        case "AfricanSwallow":  
            return new AfricanSwallow(data);  
        case "NorwegianBlueParrot":  
            return new NorwegianBlueParrot(data);  
        default:  
            return new Bird(data);  
    }  
}
```

```

NorwegianBlueParrot airSpeed Velocity

```

## class Bird...

```
selectSpeciesDelegate(data) {  
  switch(data.type) {  
    case 'EuropeanSwallow':  
      return new EuropeanSwallowDelegate();  
    case 'AfricanSwallow':  
      return new AfricanSwallowDelegate(data);  
    case 'NorwegianBlueParrot':  
      return new NorwegianBlueParrotDelegate(data);  
    default: return null;  
  }  
}
```

```
class NorwegianBlueParrotDelegate...
```

```

constructor(data) {
    this._voltage = data.voltage;
    this._isNailed = data.isNailed;
}

get airSpeedVelocity() {
    return (this._isNailed) ? 0 : 10 + this._voltage / 10;
}

```

NorwegianBlueParrot  plumage   
 198  plumage  Bird

## class NorwegianBlueParrot...

```
get plumage() {
  return this._speciesDelegate.plumage;
}
```

```
class NorwegianBlueParrotDelegate...
```

11

```
get plumage() {
  if (this._voltage > 100) return "scorched";
  else return this._bird._plumage || "beautiful";
}

constructor(data, bird) {
  this._bird = bird;
  this._voltage = data.voltage;
  this._isNailed = data.isNailed;
}
```

## class Bird...

```
selectSpeciesDelegate(data) {
  switch(data.type) {
    case 'EuropeanSwallow':
      return new EuropeanSwallowDelegate();
    case 'AfricanSwallow':
      return new AfricanSwallowDelegate(data);
    case 'NorwegianBlueParrot':
      return new NorwegianBlueParrotDelegate(data, this);
    default: return null;
  }
}
```

plumage

## class Bird...

```
get plumage() {
  if (this._speciesDelegate)
    return this._speciesDelegate.plumage;
  else
    return this._plumage || "average";
}
```

□ □ □ □ □ □ □ □ □ □ □ □ □ □ □ □

## class Bird...

```
get plumage() {
  if (this._speciesDelegate instanceof NorwegianBlueParrotDelegate)
    return this._speciesDelegate.plumage;
  else
    return this._plumage || "average";
}
```

[illegible]

□□□□□□□□□□□□□□□□□□□□

## class Bird...

```
get plumage() {
  if (this._speciesDelegate)
    return this._speciesDelegate.plumage;
  else
    return this._plumage || "average";
}
```

## class EuropeanSwallowDelegate...

```
get plumage() {
  return this._bird._plumage || "average";
}
```

## class AfricanSwallowDelegate...

```
get plumage() {
  return this._bird._plumage || "average";
}
```

SpeciesDelegate plumage 属性にアクセスする代わりに、\_bird 属性にアクセスして、その plumage 属性にアクセスする。

SpeciesDelegate 375 行のコードで定義されている。

```
class SpeciesDelegate {
  constructor(data, bird) {
    this._bird = bird;
  }
  get plumage() {
    return this._bird._plumage || "average";
  }
}

class EuropeanSwallowDelegate extends SpeciesDelegate {
  // ...
}

class AfricanSwallowDelegate extends SpeciesDelegate {
  constructor(data, bird) {
    super(data, bird);
    this._numberOfCoconuts = data.numberOfCoconuts;
  }
  // ...
}

class NorwegianBlueParrotDelegate extends SpeciesDelegate {
  constructor(data, bird) {
    super(data, bird);
    this._voltage = data.voltage;
    this._isNailed = data.isNailed;
  }
  // ...
}
```

SpeciesDelegate クラスは、Bird クラスの plumage 属性にアクセスするために、SpeciesDelegate クラスの plumage 属性にアクセスする。

## class Bird...

## class SpeciesDelegate...

```

class Bird {
    SpeciesDelegate
}

Bird

```

```
function createBird(data) {
    return new Bird(data);
}

class Bird {
    constructor(data) {
        this._name = data.name;
        this._plumage = data.plumage;
        this._speciesDelegate = this.selectSpeciesDelegate(data);
    }
    get name() {return this._name;}
    get plumage() {return this._speciesDelegate.plumage;}
    get airSpeedVelocity() {return this._speciesDelegate.airSpeedVelocity;}

    selectSpeciesDelegate(data) {
        switch(data.type) {
            case 'EuropeanSwallow':
                return new EuropeanSwallowDelegate(data, this);
            case 'AfricanSwallow':
                return new AfricanSwallowDelegate(data, this);
            case 'NorwegianBlueParrot':
                return new NorwegianBlueParrotDelegate(data, this);
            default: return new SpeciesDelegate(data, this);
        }
    }
    // rest of bird's code...
}

class SpeciesDelegate {
    constructor(data, bird) {
        this._bird = bird;
    }
    get plumage() {
        return this._bird._plumage || "average";
    }
    get airSpeedVelocity() {return null;}
}

class EuropeanSwallowDelegate extends SpeciesDelegate {
    get airSpeedVelocity() {return 35;}
}

class AfricanSwallowDelegate extends SpeciesDelegate {
    constructor(data, bird) {
        super(data, bird);
        this._numberOfCoconuts = data.numberOfCoconuts;
    }
    get airSpeedVelocity() {
        return 40 - 2 * this._numberOfCoconuts;
    }
}

class NorwegianBlueParrotDelegate extends SpeciesDelegate {
    constructor(data, bird) {
        super(data, bird);
        this._voltage = data.voltage;
        this._isNailed = data.isNailed;
    }
    get airSpeedVelocity() {
        return (this._isNailed) ? 0 : 10 + this._voltage / 10;
    }
    get plumage() {
        if (this._voltage > 100) return "scorched";
        else return this._bird._plumage || "beautiful";
    }
}
```

“我”——

## Replace Inheritance with Delegation

11

11



1. 在 `CatalogItem` 类中，添加一个 `hasTag` 方法，用于检查物品是否具有指定的标签。  
 2. 在 `Scroll` 类中，添加一个 `needsCleaning` 方法，用于检查物品是否需要清洁。  
 3. 在 `Scroll` 类中，添加一个 `daysSinceLastCleaning` 方法，用于计算物品距离上次清洁的天数。  
 4. 在 `Scroll` 类中，添加一个 `dateLastCleaned` 属性，用于记录物品上次清洁的时间。

11

1. 在 `Scroll` 类中，添加一个 `needsCleaning` 方法，用于检查物品是否需要清洁。  
 2. 在 `Scroll` 类中，添加一个 `daysSinceLastCleaning` 方法，用于计算物品距离上次清洁的天数。  
 3. 在 `Scroll` 类中，添加一个 `dateLastCleaned` 属性，用于记录物品上次清洁的时间。

## class CatalogItem...

```

constructor(id, title, tags) {
  this._id = id;
  this._title = title;
  this._tags = tags;
}

get id() {return this._id;}
get title() {return this._title;}
hasTag(arg) {return this._tags.includes(arg);}
    
```

1. 在 `Scroll` 类中，添加一个 `needsCleaning` 方法，用于检查物品是否需要清洁。  
 2. 在 `Scroll` 类中，添加一个 `daysSinceLastCleaning` 方法，用于计算物品距离上次清洁的天数。  
 3. 在 `Scroll` 类中，添加一个 `dateLastCleaned` 属性，用于记录物品上次清洁的时间。

## class Scroll extends CatalogItem...

```

constructor(id, title, tags, dateLastCleaned) {
  super(id, title, tags);
  this._lastCleaned = dateLastCleaned;
}

needsCleaning(targetDate) {
  const threshold = this.hasTag("revered") ? 700 : 1500;
  return this.daysSinceLastCleaning(targetDate) > threshold ;
}

daysSinceLastCleaning(targetDate) {
  return this._lastCleaned.until(targetDate, ChronoUnit.DAYS);
}
    
```

1. 在 `Scroll` 类中，添加一个 `needsCleaning` 方法，用于检查物品是否需要清洁。  
 2. 在 `Scroll` 类中，添加一个 `daysSinceLastCleaning` 方法，用于计算物品距离上次清洁的天数。  
 3. 在 `Scroll` 类中，添加一个 `dateLastCleaned` 属性，用于记录物品上次清洁的时间。

1. 在 `Scroll` 类中，添加一个 `needsCleaning` 方法，用于检查物品是否需要清洁。  
 2. 在 `Scroll` 类中，添加一个 `daysSinceLastCleaning` 方法，用于计算物品距离上次清洁的天数。  
 3. 在 `Scroll` 类中，添加一个 `dateLastCleaned` 属性，用于记录物品上次清洁的时间。

1. 在 `Scroll` 类中，添加一个 `needsCleaning` 方法，用于检查物品是否需要清洁。  
 2. 在 `Scroll` 类中，添加一个 `daysSinceLastCleaning` 方法，用于计算物品距离上次清洁的天数。  
 3. 在 `Scroll` 类中，添加一个 `dateLastCleaned` 属性，用于记录物品上次清洁的时间。

1. 在 `Scroll` 类中，添加一个 `needsCleaning` 方法，用于检查物品是否需要清洁。  
 2. 在 `Scroll` 类中，添加一个 `daysSinceLastCleaning` 方法，用于计算物品距离上次清洁的天数。  
 3. 在 `Scroll` 类中，添加一个 `dateLastCleaned` 属性，用于记录物品上次清洁的时间。

## class Scroll extends CatalogItem...

□ □

## Scroll CatalogItem

□ □

0000000000000000000025600000000000000000000000000000Scroll 0000  
CatalogItem 00 id 00000000 ID0000000 CatalogItem 00000000000000000000  
ID 00000000 ID00000000 ID0000000000 Scroll 0000 id 0000000 Scroll 0000000  
00000000000000000000 CatalogItem 00 id 0000000000000000000000

□□□□...

11

```
const scrolls = aDocument
  .map(record => new Scroll(record.id,
    record.catalogData.title,
    record.catalogData.tags,
    LocalDate.parse(record.lastCleaned)));
```

000000000000256000000000000000000000repository0000000000000000  
 000000000000000000CatalogItem 00000 ID 000000000000000000 ID 00  
 0 Scroll 000  
 0000000000000000001240000000000000 ID 00000000 Scroll 000000

□□□□...

```
const scrolls = aDocument
  .map(record => new Scroll(record.id,
    record.catalogData.title,
    record.catalogData.tags,
    LocalDate.parse(record.lastCleaned),
    record.catalogData.id,
    catalog));
```

## class Scroll...

```

    constructor(id, title, tags, dateLastCleaned, catalogID, catalog) {
        this._id = id;
        this._catalogItem = new CatalogItem(null, title, tags);
        this._lastCleaned = dateLastCleaned;
    }
}

```

```

Scroll catalogID CatalogItem
CatalogItem

```

## class Scroll...

```
constructor(id, title, tags, dateLastCleaned, catalogID, catalog) {
  this._id = id;
  this._catalogItem = catalog.get(catalogID);
  this._lastCleaned = dateLastCleaned;
}
```

Scroll  title  tags 124

□ □ □ □ ...

```
const scrolls = aDocument
  .map(record => new Scroll(record.id,
    record.catalogData.title,
    record.catalogData.tags,
    LocalDate.parse(record.lastCleaned),
    record.catalogData.id,
    catalog));
```

## class Scroll...

```
    constructor(id, title, tags, dateLastCleaned, catalogID, catalog) {  
    this._id = id;  
    this._catalogItem = catalog.get(catalogID);  
    this._lastCleaned = dateLastCleaned;  
    }
```