Refactoring

09 ACM

Xiao Jia

Outline

- Introduction to refactoring
- Bad smells in code
- Composing methods
- Organizing data
- Simplifying conditional expressions

Refactoring (重构)

- Refactoring is a disciplined technique for restructuring an existing body of code, altering its internal structure without changing its external behavior.

- 对内部结构的修改
- 不改变外部行为

Kent Beck's metaphor of two hats

- Adding function
  - Shouldn't be changing existing code
  - Just add new capabilities
- Refactoring
  - Not adding functions
  - Only restructure the code
- As you develop software, you probably find yourself swapping hats frequently.

Why should you refactor?

- Improve the design
  - Code changing, structure losing (cumulative effect)
  - Poor design takes more code (literally duplicated)
  - Ensure the code says everything once and only once (eliminating the duplicates)
- Make it easier to understand
- Find bugs
- Program faster

Why should you refactor?

- Improve the design
- Make it easier to understand
  - Make your code **readable** 可读性
  - Say exactly what you mean
  - Understand unfamiliar code
- Find bugs
- Program faster

Why should you refactor?

- Improve the design
- Make it easier to understand
- Find bugs
  - Understand what the code does
  - Help you **spot** bugs
  - Write **robust** code 鲁棒性、健壮性
- Program faster

Why should you refactor?

- Improve the design
- Make it easier to understand
- Find bugs
- Program faster
  - A good design is essential for rapid development.
  - *Only good designs allow rapid development.*
  - Refactoring stops the design of the system from decaying (衰败).

When should you refactor?

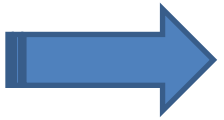
- Three strikes and you refactor.
  - The first time you do something, you just do it.
  - The second time you do something similar, you wince at the duplication, but you do the duplicate thing anyway.
  - The third time you do something similar, you refactor.

Bad smells in code

- **Duplicated code**
- **Magic numbers**
- **Long method and comments**
- Divergent change (发散型变化)
- Shotgun surgery (霰弹型修改)
- Primitive obsession (基本类型偏执)
- Speculative generality (纯臆测泛化)

Magic numbers

```
public function isExamTest()
{
    return in_array(
        $this->status,
        range(0, 2)
    );
}
```

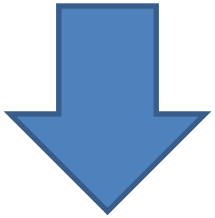


Magic numbers

```
public function isExamTest()
{
    return in_array(
        $this->status,
        array(
            self::STATUS_PENDING,
            self::STATUS_WAITING,
            self::STATUS_DONE
        )
    );
}
```

Replace Magic Number with Symbolic Constant

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

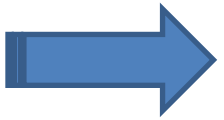


```
double const GRAVITATIONAL_CONSTANT = 9.81;
double potentialEnergy(double mass, double height) {
    return mass * GRAVITATIONAL_CONSTANT * height;
}
```

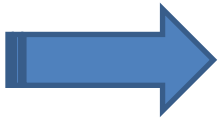
Long method and comments

- Be **aggressive** about decomposing methods
- Whenever you feel the need to comment something, **write a method instead**.
  - named after the intention of the code rather than how it does it

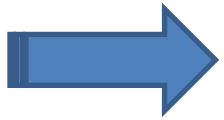
- Equivalent smell:
  too many temporary variables



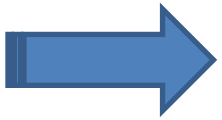
```
// read the adjacency matrix
for (int i = 1; i <= n; ++i)
  for (int j = 1; j <= n; ++j) cin >> a[i][j];
// Floyd-Warshall algorithm
for (int k = 1; k <= n; ++k)
  for (int i = 1; i <= n; ++i)
     for (int j = 1; j <= n; ++j)
          if (a[i][j] > a[i][k] + a[k][j])
              a[i][j] = a[i][k] + a[k][j];
// write the adjacency matrix
for (int i = 1; i <= n; ++i) {
  for (int j = 1; j <= n; ++j)
     cout << a[i][j] << ' ';
  cout << endl;
}
```



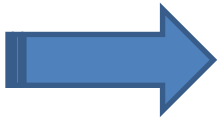
```
void readMatrix() {
  for (int i = 1; i <= n; ++i)
     for (int j = 1; j <= n; ++j)
         cin >> a[i][j];
}

void writeMatrix() {
  for (int i = 1; i <= n; ++i) {
     for (int j = 1; j <= n; ++j)
         cout << a[i][j] << ' ';
     cout << endl;
  }
}
```



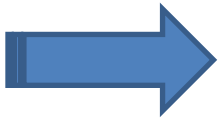
```
void algorithmFloydWarshall() {
  for (int k = 1; k <= n; ++k)
     for (int i = 1; i <= n; ++i)
          for (int j = 1; j <= n; ++j)
               if (a[i][j] > a[i][k] + a[k][j])
                    a[i][j] = a[i][k] + a[k][j];
}

int main() {
  readMatrix();
  algorithmFloydWarshall();
  writeMatrix();
  return 0;
}
```



```
void algorithmFloydWarshall() {
  for (int k = 1; k <= n; ++k)
     updateIJ(k);
}

void updateIJ(int k) {
  for (int i = 1; i <= n; ++i)
     for (int j = 1; j <= n; ++j)
          if (a[i][j] > a[i][k] + a[k][j])
               a[i][j] = a[i][k] + a[k][j];
}
```



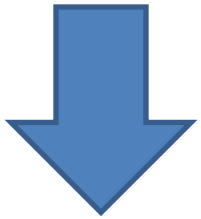
```
void algorithmFloydWarshall() {
  for (int k = 1; k <= n; ++k)
     updateIJ(k);
}

void updateIJ(int k) {
  for (int i = 1; i <= n; ++i)
   for (int j = 1; j <= n; ++j)
     updateTriangle(a[i][j], a[i][k], a[k][j]);
}

void updateTriangle(int &c, int a, int b) {
  if (c > a + b) c = a + b;
}
```



```
void shortestPath() {
  // using Floyd-Warshall algorithm
  for (int k = 1; k <= n; ++k)
     updateIJ(k);
}

void updateIJ(int k) {
  for (int i = 1; i <= n; ++i)
   for (int j = 1; j <= n; ++j)
     updateTriangle(a[i][j], a[i][k], a[k][j]);
}

void updateTriangle(int &c, int a, int b) {
  if (c > a + b) c = a + b;
}
```

Inline Temp

- ```
  int base_price = an_order.base_price;
  return base_price > 1000;
  ```



- ```
  return an_order.base_price > 1000;
  ```

- 防止局部变量污染环境
- 避免二次赋值

Replace Temp with Query

```
double price()
{
  int base_price = quantity * item_price;
  double discount_factor;
  if (base_price > 1000)
     discount_factor = 0.95;
  else
     discount_factor = 0.98;
  return base_price * discount_factor;
}
```

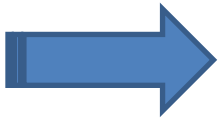
```
int base_price()
{
  return quantity * item_price;
}

int price()
{
  int a_base_price = base_price();
  double discount_factor;
  if (a_base_price > 1000)
     discount_factor = 0.95;
  else
     discount_factor = 0.98;
 return a_base_price * discount_factor;
}
```

```
int base_price()
{
  return quantity * item_price;
}

int price()
{
  double discount_factor;
  if (base_price() > 1000)
     discount_factor = 0.95;
  else
     discount_factor = 0.98;
 return base_price() * discount_factor;
}
```

```
int base_price()
{
  return quantity * item_price;
}

double discount_factor()
{
  return base_price() > 1000 ? 0.95 : 0.98;
}

int price()
{
  return base_price() * discount_factor();
}
```

Introduce Explaining Variable

```
int price()
{
   // price is base price - quantity discount + shipping
   return quantity * item_price -
         max(0, quantity - 500) * item_price * 0.05 +
         min(quantity * item_price * 0.1, 100);
}
```



```
int base_price() {
  return quantity * item_price;
}

int quantity_discount() {
  return max(0, quantity - 500) * item_price * 0.05;
}

int shipping() {
  return min(base_price() * 0.1, 100);
}

int price() {
  return base_price() - quantity_discount() + shipping();
}
```

Split Temporary Variable

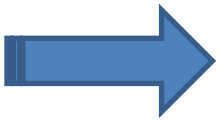
- ```
  int temp = 2 * (height + width);
  cout << temp << endl;
  temp = height * width;
  cout << temp << endl;
  ```

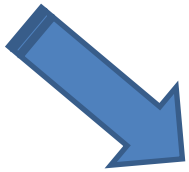
- ```
  int perimeter = 2 * (height + width);
  cout << perimeter << endl;
  int area = height * width;
  cout << area << endl;
  ```

Decompose Conditional

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

```
charge = notSummer(date) ?
    winterCharge(quantity) : summerCharge(quantity);
```

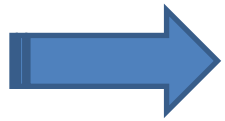




```
bool notSummer(Date date) {
    return date.before(SUMMER_START) ||
           date.after(SUMMER_END);
}

double summerCharge(int quantity) {
    return quantity * _summerRate;
}

double winterCharge(int quantity) {
    return quantity * _winterRate + _winterServiceCharge;
}
```

Consolidate Conditional Expression

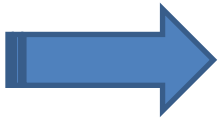
```
double disabilityAmount() {
    if (_seniority < 2) return 0;
    if (_monthsDisabled > 12) return 0;
    if (_isPartTime) return 0;
    // compute the disability amount
    ...
```

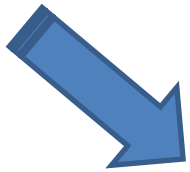


```
double disabilityAmount() {
    if (isNotEligibleForDisability()) return 0;
    // compute the disability amount
    ...
}
bool isNotEligibleForDisability() {
    return ((_seniority < 2) ||
            (_monthsDisabled > 12) ||
            (_isPartTime));
}
```

Replace Nested Conditional with Guard Clauses

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

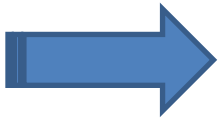


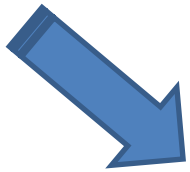


```
double getPayAmount() {
  if (_isDead) return deadAmount();
  if (_isSeparated) return separatedAmount();
  if (_isRetired) return retiredAmount();
  return normalPayAmount();
}
```

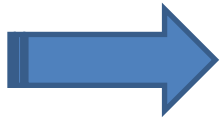
Reversing the Conditions

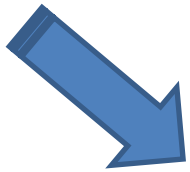
```
double getAdjustedCapital() {
  double result = 0.0;

  if (_capital > 0.0)
    if (_intRate > 0.0 && _duration > 0.0)
        result = (_income / _duration) *
                 ADJ_FACTOR;

  return result;
}
```



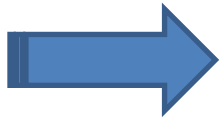


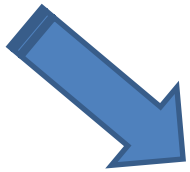
```
double getAdjustedCapital() {
  double result = 0.0;

  if (_capital <= 0.0) return result;

  if (_intRate > 0.0 && _duration > 0.0)
     result = (_income / _duration) * ADJ_FACTOR;

  return result;
}
```



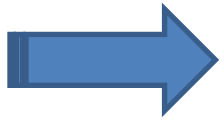


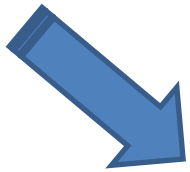
```
double getAdjustedCapital() {
  double result = 0.0;

  if (_capital <= 0.0) return result;

  if (_intRate <= 0.0 || _duration <= 0.0)
      return result;

  result = (_income / _duration) * ADJ_FACTOR;
  return result;
}
```





```
double getAdjustedCapital() {
  double result = 0.0;

  if (_capital <= 0.0) return 0.0;

  if (_intRate <= 0.0 || _duration <= 0.0)
      return 0.0;

  result = (_income / _duration) * ADJ_FACTOR;
  return result;
}
```

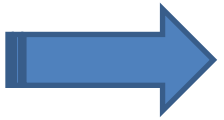


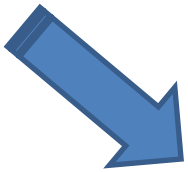


```
double getAdjustedCapital() {
  if (_capital <= 0.0) return 0.0;

  if (_intRate <= 0.0 || _duration <= 0.0)
     return 0.0;

  return (_income / _duration) * ADJ_FACTOR;
}
```

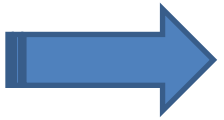
Remove Control Flag

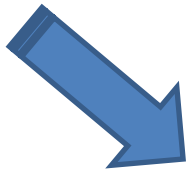
```
while (true) {
  bool cont = true;
  for (int i = 0; i < n; ++i) {
     for (int j = 0; j < n; ++j) {
        if (a[i][j] == 0) cont = false;
        if (!cont) break;
     }
     if (!cont) break;
  }
  if (!cont) break;
  // do something here
}
```



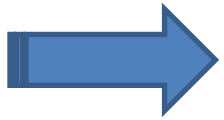


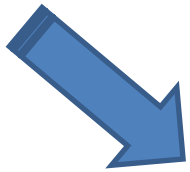
```
while (true) {
  bool cont = true;
  for (int i = 0; i < n && cont; ++i)
     for (int j = 0; j < n && cont; ++j)
         if (a[i][j] == 0) cont = false;
  if (!cont) break;
  // do something here
}
```





```
bool cont() {
  for (int i = 0; i < n; ++i)
     for (int j = 0; j < n; ++j)
        if (a[i][j] == 0) return false;
  return true;
}

while (true) {
  if (!cont()) break;
  // do something here
}
```





```
bool cont() {
  for (int i = 0; i < n; ++i)
     for (int j = 0; j < n; ++j)
         if (a[i][j] == 0) return false;
  return true;
}

while (cont()) {
  // do something here
}
```

That is not the half of it

- You need much more practice and experience
- “Code is poetry”
- Programming language is just another language (like Chinese or English)
- “Programming is about managing complexity”
- ……

Always Challenge Miracles

THANK YOU FOR LISTENING