

# 計算機科学第一

## 第二回 Readable Code

# 人々は言う

- きれいにインデントしなさい

# 人々は言う

- きれいにインデントしなさい
- きちんとコメントを書きなさい

# 人々は言う

- きれいにインデントしなさい
- きちんとコメントを書きなさい
- 一貫したスタイルで書きなさい

# 人々は言う

- きれいにインデントしなさい
- きちんとコメントを書きなさい
- 一貫したスタイルで書きなさい
- 簡潔なプログラムを書きなさい

# 人々は言う

- きれいにインデントしなさい
- きちんとコメントを書きなさい
- 一貫したスタイルで書きなさい
- 簡潔なプログラムを書きなさい
- 明快なプログラムを書きなさい

# 人々は言う

- きれいにインデントしなさい
- きちんとコメントを書きなさい
- 一貫したスタイルで書きなさい
- 簡潔なプログラムを書きなさい
- 明快なプログラムを書きなさい
- わかりやすい変数名を使いなさい

# 人々は言う

- きれいにインデントしなさい
- きちんとコメントを書きなさい
- 一貫したスタイルで書きなさい
- 簡潔なプログラムを書きなさい
- 明快なプログラムを書きなさい
- わかりやすい変数名を使いなさい
- 長いメソッドは複数のメソッドに分割しなさい



# 人々は言う

- きれいにインデントしなさい
- きちんとコメントを書きなさい
- 一貫したスタイルで書きなさい
- 簡潔なプログラムを書きなさい
- 明快なプログラムを書きなさい
- わかりやすい変数名を使いなさい
- 長いメソッドは複数のメソッドに分割しなさい
- 大きなクラスは複数のクラスに分割しなさい

# 人々は言う

- きれいにインデントしなさい
- きちんとコメントを書きなさい
- 一貫したスタイルで書きなさい
- 簡潔なプログラムを書きなさい
- 明快なプログラムを書きなさい
- わかりやすい変数名を使いなさい
- 長いメソッドは複数のメソッドに分割しなさい
- 大きなクラスは複数のクラスに分割しなさい
- 変数のスコープを小さくしなさい

# 人々は言う

- きれいにインデントしなさい
- きちんとコメントを書きなさい
- 一貫したスタイルで書きなさい
- 簡潔なプログラムを書きなさい
- 明快なプログラムを書きなさい
- わかりやすい変数名を使いなさい
- 長いメソッドは複数のメソッドに分割しなさい
- 大きなクラスは複数のクラスに分割しなさい
- 変数のスコープを小さくしなさい

いう

# 人々は言う

- きれいにインデントしなさい
- きちんとコメントを書きなさい
- 一貫したスタイルで書きなさい
- 簡潔なプログラムを書きなさい
- 明快なプログラムを書きなさい
- わかりやすい変数名を使いなさい
- 長いメソッドは複数のメソッドに分割しなさい
- 大きなクラスは複数のクラスに分割しなさい
- 変数のスコープを小さくしなさい

# 人々は言う

- きれいにインデントしなさい
- きちんとコメントを書きなさい
- 一貫したスタイルで書きなさい
- 簡潔なプログラムを書きなさい
- 明快なプログラムを書きなさい
- わかりやすい変数名を使いなさい
- 長いメソッドは複数のメソッドに分割しなさい
- 大きなクラスは複数のクラスに分割しなさい
- 変数のスコープを小さくしなさい

うる

# 人々は言う

- きれいにインデントしなさい
- きちんとコメントを書きなさい
- 一貫したスタイルで書きなさい
- 簡潔なプログラムを書きなさい
- 明快なプログラムを書きなさい
- わかりやすい変数名を使いなさい
- 長いメソッドは複数のメソッドに分割しなさい
- 大きなクラスは複数のクラスに分割しなさい
- 変数のスコープを小さくしなさい

う

る

さ

# 人々は言う

- きれいにインデントしなさい
- きちんとコメントを書きなさい
- 一貫したスタイルで書きなさい
- 簡潔なプログラムを書きなさい
- 明快なプログラムを書きなさい
- わかりやすい変数名を使いなさい
- 長いメソッドは複数のメソッドに分割しなさい
- 大きなクラスは複数のクラスに分割しなさい
- 変数のスコープを小さくしなさい

う

る

さ

い

# 本当に大事なことは

- きれいにインデントしなさい
- きちんとコメントを書きなさい
- 一貫したスタイルで書きなさい
- 簡潔なプログラムを書きなさい
- 明快なプログラムを書きなさい
- わかりやすい変数名を使いなさい
- 長いメソッドは複数のメソッドに分割しなさい
- 大きなクラスは複数のクラスに分割しなさい
- 変数のスコープを小さくしなさい



# 本当に重要なことは

- きれいにインデントしなさい
- きちんとコメントを書きなさい

ではなくて

- 学習したスタイルで書きなさい
- 簡潔なプログラムを書きなさい
- 明快なプログラムを書きなさい
- わかりやすい変数名を使いなさい
- 長いメソッドは複数のメソッドに分割しなさい
- 大きなクラスは複数のクラスに分割しなさい
- 変数のスコープを小さくしなさい

# 本当に重要なことは

- きれいにインデントしなさい
- きちんとコメントを書きなさい

ただ、ひとつ

- 一貫したスタイルで書きなさい
- 簡潔なプログラムを書きなさい
- 明瞭なプログラムを書きなさい
- わかりやすい変数名を使いなさい
- 長いメソッドは複数のメソッドに分割しなさい
- 大きなクラスは複数のクラスに分割しなさい
- 変数のスコープを小さくしなさい

# 本当に重要なことは

良いプログラムを書きなさい

# 本当に重要なことは

**良い**プログラムを書きなさい

# 本当に重要なことは

**良い**プログラムを書きなさい

# 本当に重要なことは

**良い**プログラムを書きなさい

# 本当に重要なことは

**良い**プログラムを書きなさい

# 本当に重要なことは

**良い**プログラムを書きなさい



本当に重要なことは

良い

プログラムを書きなさい

良いプログラム？

# 良いプログラム

- バグが少ない
- 簡潔明瞭
- 理解しやすい
- 拡張しやすい
- バグを修正しやすい

# 良いプログラム

- バグが少ない
- 簡潔明瞭
- 理解しやすい
- 拡張しやすい
- バグを修正しやすい

でも、良いプログラムを書くには長年の経験と勘が必要じゃないの？

# プログラムの**可読性**原理

プログラムは、自分以外の誰かが読んで  
理解する時間を最小化すべきだ

# プログラムの**可読性**原理

プログラムは、**自分以外の誰か**が読んで  
理解する時間を最小化すべきだ

このコードは誰にも見せないもん

# プログラムの**可読性**原理

プログラムは、**自分以外の誰か**が読んで  
理解する時間を最小化すべきだ

このコードは誰にも見せないもん  
⇒ **諸君のコードは僕やTAが見る**

# プログラムの**可読性**原理

プログラムは、**自分以外の誰か**が読んで  
理解する時間を最小化すべきだ

このコードは誰にも見せないもん

⇒ **諸君が会社で書くコードはプロジェクトの他人が見る**



# プログラムの**可読性**原理

プログラムは、**自分以外の誰か**が読んで  
理解する時間を最小化すべきだ

このコードは誰にも見せないもん

⇒ **ひと月前の自分は赤の他人。嘘だと  
思ったら、夏学期に書いたコードを見直  
してごらん。**

# プログラムの**可読性**原理

プログラムは、自分以外の**誰かが読んで理解**する時間を最小化すべきだ

このプログラムのどこが分りにくいって  
言うのさ？

# プログラムの**可読性**原理

プログラムは、自分以外の**誰かが読んで理解**する時間を最小化すべきだ

このプログラムのどこが分りにくいって言うのさ？

プログラムの前提知識を持たない人が読むことを想定しなさい。

# プログラムの**可読性**原理

プログラムは、自分以外の誰かが読んで  
**理解する**時間を最小化すべきだ

これ見れば大体わかるでしょ？

# プログラムの**可読性**原理

プログラムは、自分以外の誰かが読んで  
**理解する**時間を最小化すべきだ

これ見れば大体わかるでしょ？

いい加減な理解ではなく、プログラムの  
正当性の検証、機能追加、バグ修正がで  
きるレベルの理解が可能なことが重要。

# プログラムの**可読性**原理

プログラムは、自分以外の誰かが読んで  
**理解する**時間を最小化すべきだ

これ見れば大体わかるでしょ？      **読めるプログラム ⇒**  
いい加減な理解ではなく、プログラムの **良いプログラム**  
正当性の検証、機能追加、バグ修正がで  
きるレベルの理解が可能なことが重要。

# プログラムの**可読性**原理

プログラムは、自分以外の誰かが読んで  
理解する**時間を最小化**すべきだ

じゃ、徹底的に短いコードを書けばいい  
んだね！

# プログラムの**可読性**原理

プログラムは、自分以外の誰かが読んで  
**理解する時間を最小化**すべきだ

じゃ、徹底的に短いコードを書けばいい  
んだね！

そういう傾向はあるけれど、短くするばかりに分りにくくしてはいけない。



# 短ければいいのか？

- RightMostBit.java を参照
  - 3種類のコードのいずれがよいか？

# プログラムの**可読性**原理

プログラムは、自分以外の誰かが読んで  
**理解する**時間を最小化すべきだ

可読性を向上すれば、

あなたは有能になった気分になるし、

バグも減って、

みんながあなたのコードを使いたがる

# きれいなコードの基礎

- 識別子の名前に情報を詰め込もう
- 誤解を避ける命名を試みよう
- 適切なコメントを書こう
- 変数のスコープに気をつけよう
- 一般的な命名規則にしたがおう

# 識別子の名前に 情報を詰め込むこと

- 意味を持たない名前の識別子を安易に使ってはいけない
  - tmp
  - retval
  - foo, bar, baz, qux
  - hoge, piyo, fuga, boge, poi
  - a, b, c, ...

え～、でも名前が思いつかないよ～

- 変数の値を説明する言葉や変数を利用する目的を名前にしましょう

え～、でも名前が思いつかないよ～

```
double euclideanNorm(double[] v) {  
    var retval = 0.0;  
    for (int i = 0; i < v.length; i++) {  
        retval += v[i]*v[i];  
    }  
    return Math.sqrt(retval);  
}
```

え～、でも名前が思いつかないよ～

```
double euclideanNorm(double[] vector) {  
    var retval = 0.0;  
    for (int i = 0; i < vector.length; i++) {  
        retval += vector[i]*vector[i];  
    }  
    return Math.sqrt(retval);  
}
```

え～、でも名前が思いつかないよ～

```
double euclideanNorm(double[] vector) {  
    var sum_squares = 0.0;  
    for (int i = 0; i < vector.length; i++) {  
        sum_squares += vector[i]*vector[i];  
    }  
    return Math.sqrt(sum_squares);  
}
```



さらに、

```
double euclideanNorm(double[] vector) {  
    var sum_squares = 0.0;  
    for (double v : vector)  
        sum_squares += v * v;  
    return Math.sqrt(sum_squares);  
}
```

# tmpが許されることも

```
{  
    // compare and swap if necessary  
    if (large < small) {  
        int tmp = large;  
        large = small;  
        small = tmp;  
    }  
}
```

# Javaに並列代入があったら？

```
{  
    // compare and swap if necessary  
    if (large < small)  
        (small, large) = (large, small);  
}
```

# 典型的な手抜き の例

```
{  
    String tmp = user.name();  
    tmp += " " + user.phone_number();  
    tmp += " " + user.email();  
    template.set("user_info", tmp);  
}
```

# 典型的な手抜き の例

```
{  
    String user_info = user.name();  
    user_info += " " + user.phone_number();  
    user_info += " " + user.email();  
    template.set("user_info", user_info);  
}
```

# tmpももうひと工夫できる例

```
{  
    File tmp = File.createTempFile("tmp-", ".txt");  
  
    ...  
  
    saveData(tmp);  
}
```

# tmpももうひと工夫できる例

```
{  
    File tmp_file = File.createTempFile("tmp-", ".txt");  
  
    ...  
  
    saveData(tmp_file);  
}
```

# tmpももうひと工夫できる例

```
{  
    File.createTempFile("cache-", ".txt");  
  
    ...  
  
    saveData(tmp_file);  
}
```



# ループ変数

- 慣習的にループ変数に `i, j, iter, it` などが使われる。これは構わない。
- 逆に、ループ変数ではないものに `i, j, iter, it` と命名すると混乱する。  
やめましょう。

とはいえ、

```
for (int i = 0; i < clubs.size(); i++)  
    for (int j = 0; j < clubs[i].members.size(); j++)  
        for (int k = 0; k < users.size(); k++)  
            if (clubs[i].members[k] == users[j])  
                out.printf("user[%d] is in club[%d]\n", j, i);
```

**このプログラムのバグを言いあててください**

とはいえ、

```
for (int i = 0; i < clubs.size(); i++)  
    for (int j = 0; j < clubs[i].members.size(); j++)  
        for (int k = 0; k < users.size(); k++)  
            if (clubs[i].members[k] == users[j])  
                out.printf("user[%d] is in club[%d]\n", j, i);
```

**このプログラムのバグを言いあててください**

# とはいえ、


```
for (int i = 0; i < clubs.size(); i++)  
  for (int j = 0; j < clubs[i].members.size(); j++)  
    for (int k = 0; k < users.size(); k++)
```

集合	ループ変数
clubs	i
club members	j
users	k

# わかりにくい対応

```
for (int i = 0; i < clubs.size(); i++)  
  for (int j = 0; j < clubs[i].members.size(); j++)  
    for (int k = 0; k < users.size(); k++)
```

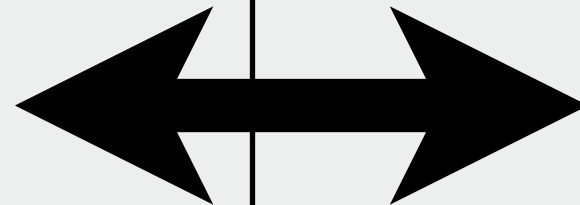
集合	ループ変数
clubs	i
club members	j
users	k



# こうしてみるのも一手

```
for (int i = 0; i < clubs.size(); i++)  
  for (int j = 0; j < clubs[i].members.size(); j++)  
    for (int k = 0; k < users.size(); k++)
```

集合	ループ変数
clubs	club_i
club members	member_i
users	user_i



# こうしてみるのも一手

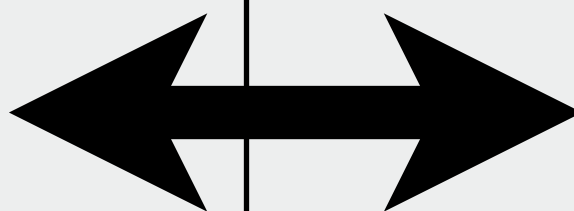
```
for (int club_i = 0; club_i < clubs.size(); club_i++)  
    for (int member_i = 0;  
         member_i < clubs[club_i].members.size(); member_i++)  
        for (int user_i = 0;  
             user_i < users.size(); user_i++)  
            if (clubs[club_i].members[member_i] == users[user_i])  
                out.printf("user[%d] is in club[%d]\n", user_i, club_i);
```

ちょっと煩雑かなあ。。。。

# では、こうしてみるか

```
for (int ci = 0; ci < clubs.size(); ci++)  
    for (int mi = 0; mi < clubs[i].members.size(); mi++)  
        for (int ui = 0; ui < users.size(); ui++)
```

集合	ループ変数
clubs	ci
club members	mi
users	ui





# こうしてみるのも一手

```
for (int ci = 0; ci < clubs.size(); ci++)  
    for (int mi = 0; mi < clubs[ci].members.size(); mi++)  
        for (int ui = 0; ui < users.size(); ui++)  
            if (clubs[ci].members[mi] == users[ui])  
                out.printf("user[%d] is in club[%d]\n", ui, ci);
```

簡素だけど、対応はわかりやすい

# LISPやSchemeなら

```
for (club : club*)
```

```
...
```

のような感じに \* をつけて集合を表す識別子、  
\*がないと要素を表す識別子と書き分けられる

# きれいなコードの基礎

- 識別子の名前に情報を詰め込もう
- 誤解を避ける命名を試みよう
- 適切なコメントを書こう
- 変数のスコープに気をつけよう
- 一般的な命名規則にしたがおう

# 識別子に単位を含める

いまいち	もっと良い
<code>Thread.sleep(long t)</code>	<code>Thread.sleep(long ms)</code>
<code>newFile(int size)</code>	<code>newFile(int size_mb)</code>
<code>throttleDownload(float limit)</code>	<code>throttleDownload(float max_kbps)</code>
<code>rotate(float angle)</code>	<code>rotate(float radian)/ rotate(float degree)</code>
<code>deposit(long amount)</code>	<code>depositDollar(long amount)</code>

# きれいなコードの基礎

- 識別子の名前に情報を詰め込もう
- 誤解を避ける命名を試みよう
- 適切なコメントを書こう
- 変数のスコープに気をつけよう
- 一般的な命名規則にしたがおう

# コメントのためのコメント

```
class Complex {  
    // フィールド  
    double re, im;  
    // コンストラクタ  
    public Complex(double re, double im) ...  
    // addメソッド  
    public Complex add(Complex c) ...  
    // toString メソッド  
    public String toString() ...  
}
```

意味がないコメントは無駄にプログラムの  
行数を増し、理解を困難にする

# コメントのためのコメント

```
class Complex {  
    double re, im;  
    public Complex(double re, double im) ...  
    public Complex add(Complex c) ...  
    public String toString() ...  
}
```

無駄なコメントはむしろ削除しましょう

# コメントのためのコメント

```
/* 与えられた木から、名前と深さが合致するものを見つける */
```

```
Node findNodeInTree(Node tree, String name, int depth) { ...
```

メソッド名で言い尽された内容で無駄  
逆にメソッド名は工夫されている



# コメントのためのコメント

```
/* 木の根から深さがdepth以内でnameに合致するノードを  
   見つける。複数あるなら最初に見つけたもの。見つからない  
   場合には null */
```

```
Node findNodeInTree(Node tree, String name, int depth) { ...
```

メソッド名で表現されない内容が有用。

動作の詳細、境界条件。

# 長い名前は嫌？

- 昔の人ならね、
- Eclipseの便利ツールを使いこなそう
  - 名前の自動補完 (Option /) Windowsなら  
Meta /
  - Renameツール

# Eclipseの便利機能

働き	Mac	Windows
名前の補完	Option /	Alt /
名前の一括変更	Option Command R	Alt Shift R

# プログラムの**可読性**原理

プログラムは、自分以外の誰かが読んで  
理解する時間を最小化すべきだ