

# 計算機科學第一 (1,2-可読性原理)

脇田建 (<http://kwakita.wordpress.com/classes/classes-2013/cs1/>)

---

日付

人々は言う

---

# 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

# 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

# でも

---

# 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
- ＊ 貫したスタイルで書きなさい
- ＊ 純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

# でも

# うる

- 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
- 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

# でも

---

うるさく

- 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

- 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

でも

---

うるさい

ささい

- 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

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
- ＊ 一貫したスタイルを書きなさい
- ＊ 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

ではなくて

# 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
- ✿ 一貫したスタイルで書きなさい
- ✿ 漢字プログラマーを書きなさい
- ✿ 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

た  
だ  
い

ひ  
と  
つ

# 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？

---

- ❖ 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

⇒ 会社でのコードはプロジェクトメンバ  
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

⇒ **ひと月後の自分**は赤の他人。

嘘だと思ったら、前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くて  
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くらい  
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できることが重要。

# プログラムの**可読性**原理

---

**読めるプログラム ⇒ 良いプログラム**

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できることが重要。

# 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

じや、徹底的に短いコードを書けばいい

んだ！

# 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

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。

でも、分り易さを犠牲にしてはいけない。

# 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 tmp = 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、

---

```
class Club { List<String> members = new ArrayList<String>(); }
List<Club> clubs = new ArrayList<Club>();
List<String> users = new ArrayList<String>();

void printMembershipBug() {
    for (int i = 0; i < clubs.size(); i++)
        for (int j = 0; j < clubs.get(i).members.size(); j++)
            for (int k = 0; k < users.size(); k++)
                if (clubs.get(i).members.get(k) == users.get(j))
                    System.out.printf("User[%d] belongs to club %d.", j, i);
}
```

**このプログラムのバグを言いあてて下さい**

とはいえ、

---

```
class Club { List<String> members = new ArrayList<String>(); }
List<Club> clubs = new ArrayList<Club>();
List<String> users = new ArrayList<String>();
```

```
void printMembershipBug() {
    for (int i = 0; i < clubs.size(); i++)
        for (int j = 0; j < clubs.get(i).members.size(); j++)
            for (int k = 0; k < users.size(); k++)
                if (clubs.get(i).members.get(k) == users.get(j))
                    System.out.printf("User[%d] belongs to club %d.", j, i);
}
```

**ちょっとした添字の書き間違いというのは嫌らしいエラー**

とはいえ、

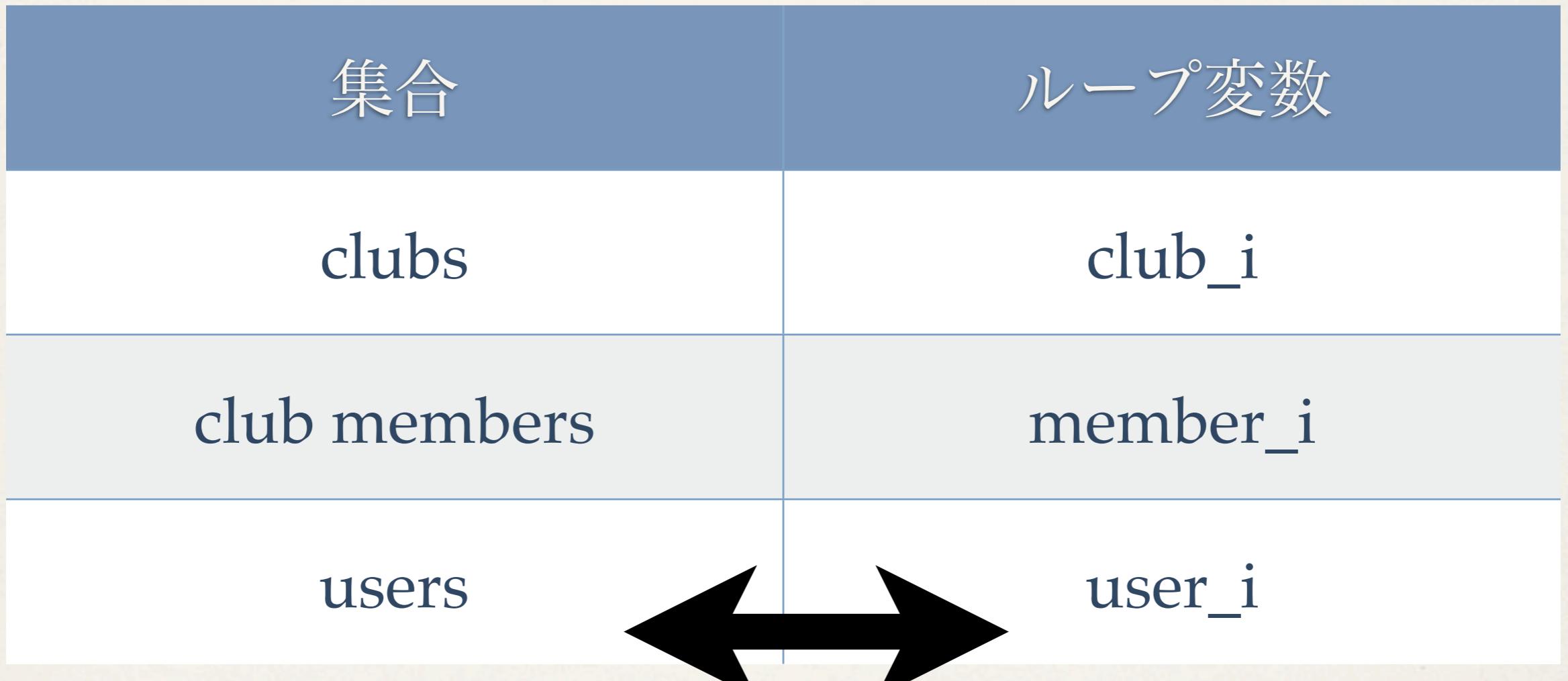
---

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

# わかりにくいくい対応



# こうしてみるのも一手



# こうしてみるのも一手

```
for (int club_i = 0; club_i < clubs.size(); club_i++)  
    for (int member_j = 0; member_j < clubs.get(club_i).members.size();  
        member_j++)  
        for (int user_k = 0; user_k < users.size(); user_k++)  
            if (clubs.get(club_i).members.get(user_k) == users.get(member_j))  
                System.out.printf("User[%d] belongs to club %d.",  
                    member_j, club_i);
```

ちょっと煩雑かなあ。。。。

# では、こうしてみるか



# こうしてみるのも一手

```
for (int ci = 0; ci < clubs.size(); ci++)  
    for (int mi = 0; mi < clubs.get(ci).members.size(); mi++)  
        for (int ui = 0; ui < users.size(); ui++)  
            if (clubs.get(ci).members.get(mi) == users.get(ui))  
                System.out.printf("User[%d] belongs to club %d.", mi, ci);
```

簡素だけど、対応はわかりやすい

# for文のかわりに forIn 文を使うと

```
for (Club club : clubs)
    for (String member : club.members)
        for (String user : users)
            if (member == user)
                System.out.printf("User[%s] belongs to club %s",
                                  user, club);
```

そもそもループ変数がいらないから、プログラムがずっと簡素

# 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

# メソッド宣言をもっとわかりやすくしたい ヒント：識別子に単位を含める

いまいち

Thread.sleep(long t)

newFile(int size)

throttleDownload(float  
limit)

rotate(float angle)

deposit(long amount)

もっと良い

Thread.sleep(long **ms**)

# 識別子に単位を含める

いまいち

Thread.sleep(long t)

newFile(int size)

throttleDownload(float  
limit)

rotate(float angle)

deposit(long amount)

もっと良い

Thread.sleep(long **ms**)

newFile(int **size\_mb**)

throttleDownload(float **max\_kbps**)

rotate(float **radian**)/  
rotate(float **degree**)

depositDollar(long **amount**)

# 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() ...  
}
```

無駄なコメントはプログラムの行数を増すだけ。

# ~~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よう

# 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)  
{ ...
```

コメントはメソッド名で言い尽された内容の繰り返し  
逆にメソッド名は工夫されている

# 有意義なコメント

---

```
/* 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