メソッド（４）

擬似コード　テスト技法

http://java.cis.k.hosei.ac.jp/

授業の前に自己点検
以下のことがらを友達に説明できますか？

- メソッドの宣言とは、起動とは何ですか
- メソッドの宣言はどのように書きますか
- メソッドの宣言はどこに置きますか
- メソッドの起動はどのようにしますか
- メソッドの仮引数、実引数、戻り値とは何ですか
- メソッドの起動にあたって実引数はどのようにして仮引数に渡されますか
- 戻り値はどのように利用しますか
- 変数のスコープとは何ですか
- 再帰呼び出しはどのように実行されますか

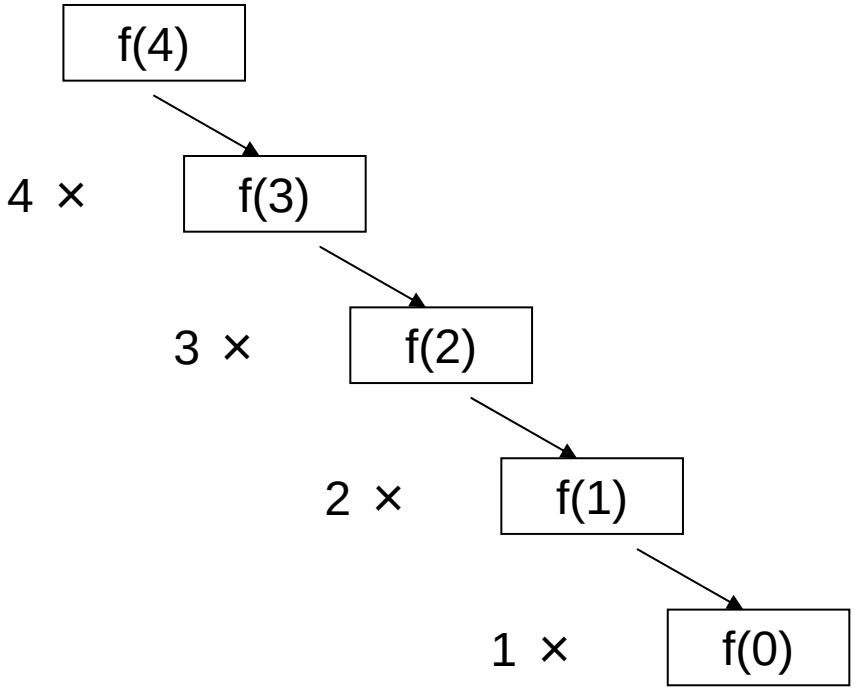
復習： 階乗

考え方

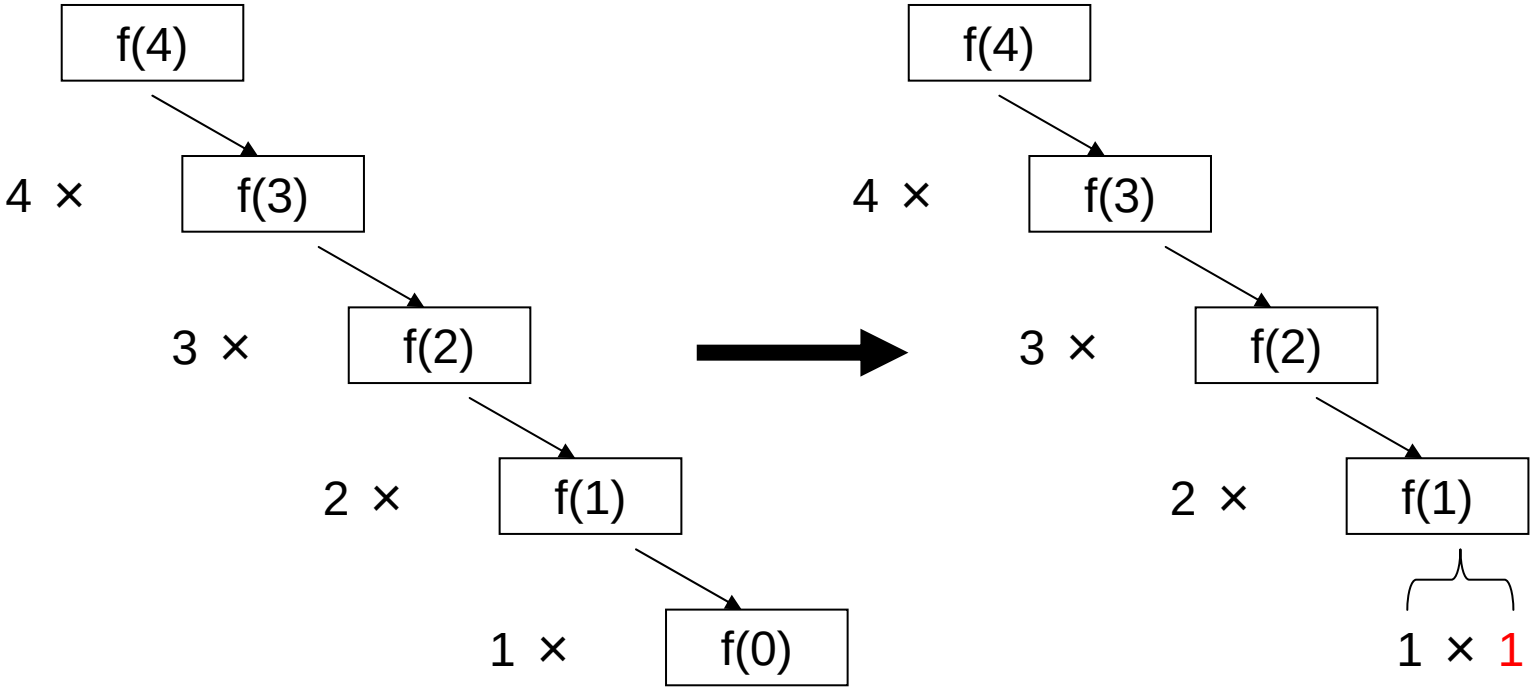
```
factorial(N) = 1                          N = 0 のとき
factorial(N) = N * factorial(N-1)  それ以外のとき
```

```
public static int factorial(int n) {
  // factorial(N) = 1 (N = 0 のとき)
  if (n == 0) {
    return 1;
  }
  // factorial(n) = n * factorial(n-1) (それ以外のとき)
  else {
    return n * factorial(n - 1);
  }
}
```

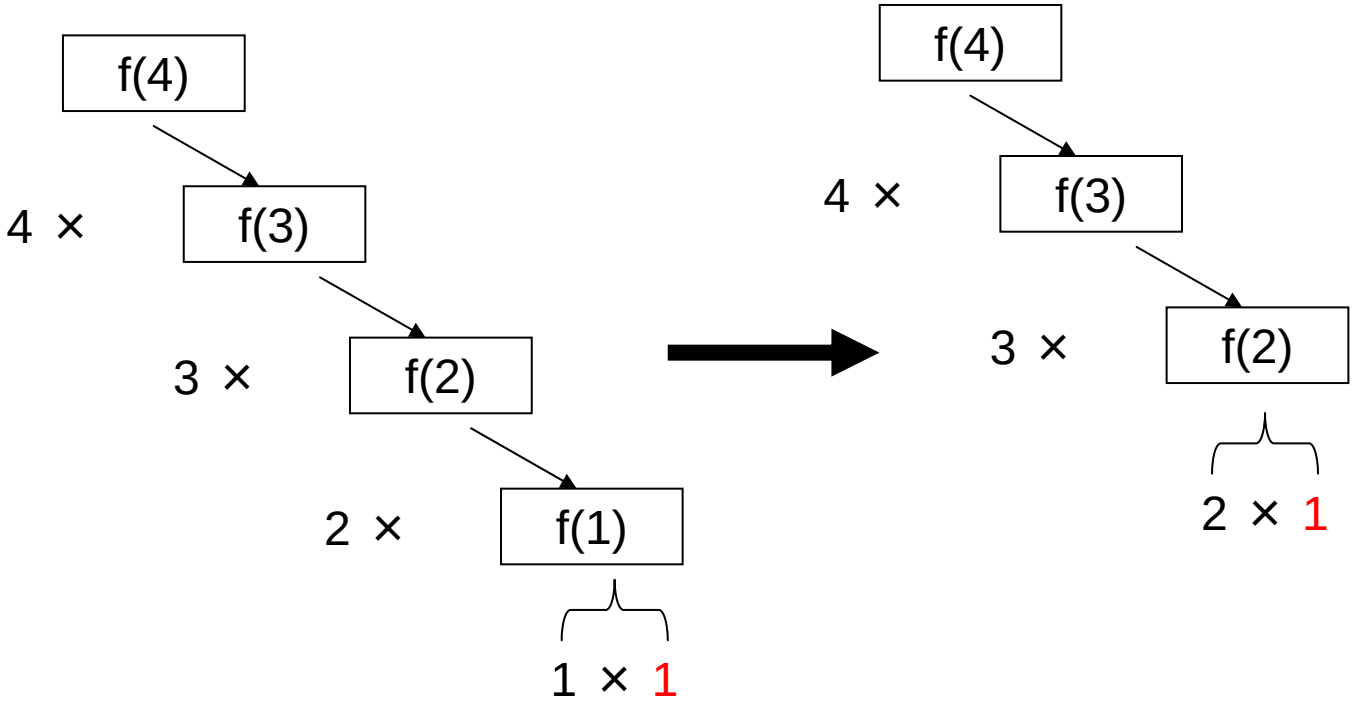
factorial(4)（以下f(4)と略記する）を起動すると、このような起動の連鎖が起こる



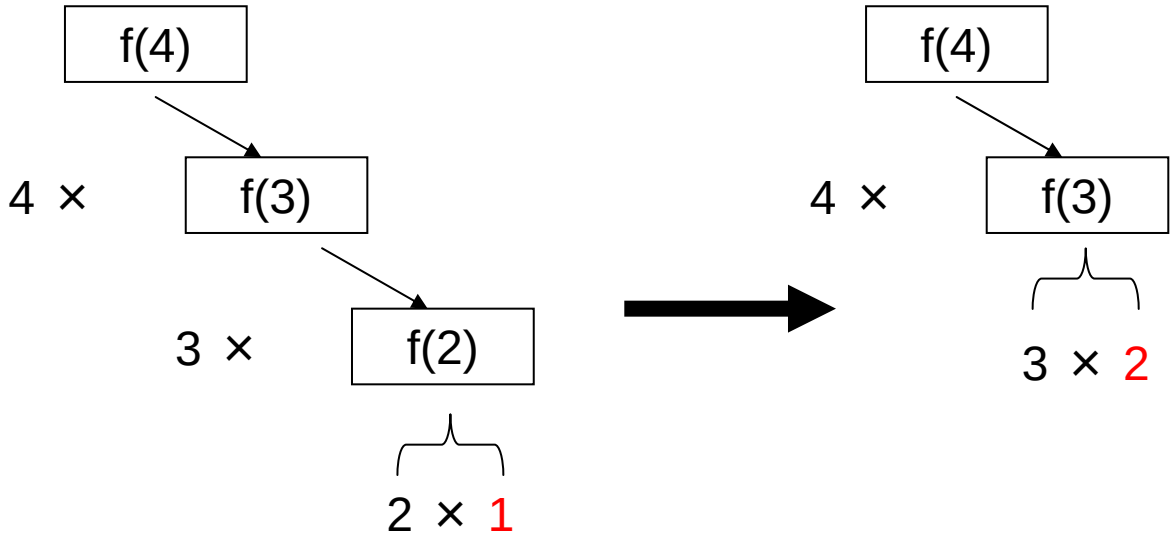
factorial(0)は即座に処理を完了して0を呼び出し側f(1)に返す



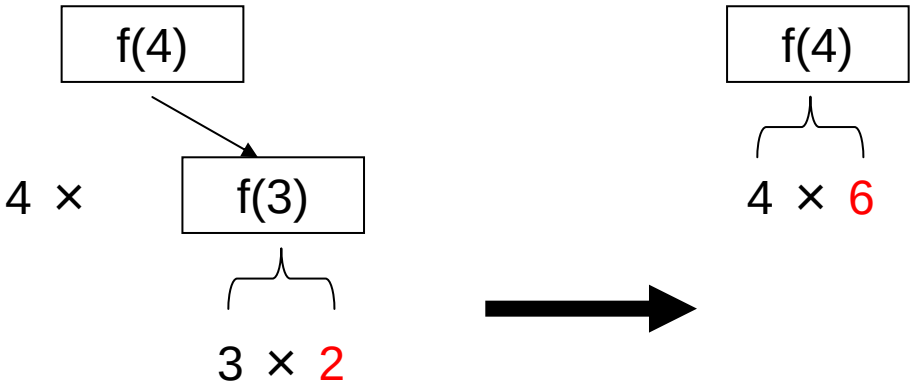
factorial(1)は掛け算を実行し、その結果1を呼び出し側f(2)に返す



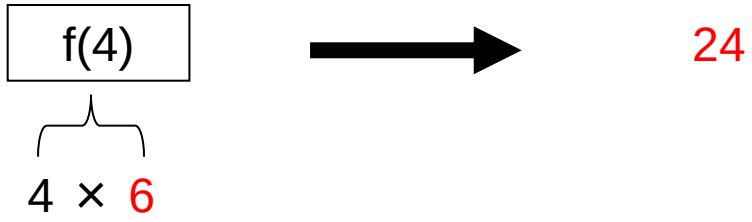
factorial(2)は掛け算を実行し、その結果2を呼び出し側f(3)に返す



factorial(3)は掛け算を実行し、その結果6を呼び出し側f(4)に返す



factorial(4)は掛け算を実行し、その結果24を呼び出し側に返す



再帰呼び出しは起動インスタンスたちの見事なチームプレイで最終結果を出す

- 個々の起動インスタンスの責任は限定されている
- factorialメソッドの個々の起動インスタンスの責任
  - 自分が末端の起動インスタンス(n==0)ならば、即座に処理を完了して1を返す
  - 自分が末端でなければ、下位の起動インスタンスから返される結果を待って、その結果にnを掛けた結果を上位の起動インスタンスに返す

今回のテーマ

- 擬似コード(pseudocode)
  - 自然言語と通常のプログラミング言語を混ぜたようなコード (プログラム)
- プログラムのテスト
  - テストの種類
  - テスト技法

擬似コード(pseudocode)

- 擬似コードとは
  - 自然言語と通常のプログラミング言語を混ぜたようなコード (プログラム)
  - アルゴリズム辞典などでよく使われる
- 目的は
  - プログラムの処理や流れを簡単に記述するものである。
  - アイデアを整理したり他人に伝えるために用いる。
- 望ましい性質
  - どのプログラミング言語にも書き直しがしやすい

擬似コードいろいろ

「入力された2つの値のうち、大きいものを返す」

```
// 擬似コード
max-of (a, b)
  if a のほうが b より大きい
    a を返す
  else
    b を返す
```

```
// 擬似コード (2) - ほとんどJava
maxOf(a,b) {
  if (a > b)
    return a
  else
    return b
}
```

```
// 擬似コード (3) - ほとんど自然言語
大きいものを返すメソッド
  入力: a, b
    もし (a のほうが b より大きい) ならば
    結果は a
  そうでなければ
    結果は b
```

Javaでの実現

「入力された2つの値のうち、大きいものを返す」

```
// 擬似コード
max-of (a, b)
  if a のほうが b より大きい
    a を返す
  else
    b を返す
```

```
// プログラミング言語(Java)
public class Max {
  public static int maxOf(int a, int b) {
    if (a > b)
      return a;
    else
      return b;
  }
}
```

擬似コードの基本

- 「コンピュータが理解できるプログラム」で表現することを目標にするのではなく「人間が理解できる作業手順」で表現することを目標とする。
- 一般的には特定のプログラミング言語にとらわれずに書く
  - Javaでの実現を目的としているならJava風でもよい
- 処理手順を中心に考えたコードを作成する
  - 各プログラミング言語に依存する、変数の型 (整数や実数など) については考えない
  - Java では int 型の値に対して割り算を行うと、小数点以下が切り捨てられる。「切り捨て」を行うことがそのプログラムの中で必須である場合、それは擬似コードの中で明示的に書いておく

擬似コードのTIPS

```
BufferedReader reader =
  new BufferedReader(new InputStreamReader(System.in));
int input1 = Integer.parseInt();
int input2 = Double.parseDouble();
```

あまりにJava的

擬似コードではこれくらいでよい

```
input1 = 入力された値 (整数)
input2 = 入力された値 (実数)
```

擬似コードのTIPS

```
System.out.println("文字列");
```

あまりにJava的

擬似コードではこれくらいでよい

```
print "文字列"
```

擬似コードの基本事項を適用してみる

「コンソールに入力された値の絶対値を表示する」

擬似コード

```
input = 入力された値
print(|input|)
```

Javaでの実現

```
package j1.lesson10;

import java.io.*;
public class Absolute {
  public static void main(String[] args) throws IOException {
    BufferedReader reader =
      new BufferedReader(new InputStreamReader(System.in));
    int input = Integer.parseInt(reader.readLine());
    System.out.println(Math.abs(input));
  }
}
```

擬似コードの作成の大まかな手順

1. 問題を分析する
2. 問題を解決する手順を考える
3. 概略レベルの擬似コードを作成する
4. 問題を解決する手順に名前をつける
5. 問題を解決する手順を詳細化する
6. 詳細レベルの擬似コードを作成する

擬似コード作成の留意点

- 処理を正確に、詳細に説明すること
- 特定のプログラミング言語に依存しない
  - ただし、分岐 (if-else) や繰り返し (for, while) などはプログラミング言語の文法を使うことが多い
- プログラムする方法 (実装手段) ではなく、プログラムにしたいこと (目的) を中心に考える
- 何らかのプログラミング言語を少し理解しているだけで読めるように書く

プログラムのテスト

- 機能テスト
  - プログラム全体に対して一連の操作を行い、プログラム全体が正しく動いているかテスト
  - プログラムが要求された仕様に沿って作られているかチェックすることができる
- 単体テスト
  - 各メソッドをいくつかのパラメータを与えて起動し、それぞれが正しく動いているかテスト
  - 各メソッドが要求された仕様に沿って作られているかチェックすることができる
- 単体テストと機能テスト
  - 一般的に、単体テスト→機能テストの順に行う
  - まずプログラムの部分部分が正しく作成できているかチェック
  - このチェックの後、機能テストでエラーを検出すると、それぞれの部分を結合した部分にエラーが含まれている可能性が高いということになる
- テストケース
  - 「実施する処理の内容」「入力データ」「期待する結果」をセットにしたもの

テストケースの例

```
public static int square(int x) {
  return x * x;
}
```

実施する処理の内容	入力データ	期待する結果
square(int)を実行	-10	100
square(int)を実行	-1	1
square(int)を実行	0	0
square(int)を実行	1	1
square(int)を実行	5	25
square(int)を実行	20	400

テスト技法　同値分割法

「同じ処理をする範囲」は入力値の同値クラス

```
public static int equiv(int n) {
  if (0 <= n && n <= 6) {
    return 120;
  } else if (7 <= n && n <= 64) {
    return 200;
  } else if (65 <= n) {
    return 120;
  } else {
    return -1;
  }
}
```

範囲	処理
~ -1	-1 を返す
0 ~ 6	120 を返す
7 ~ 64	200 を返す
65 ~	120 を返す

各同値クラスから入力値を1つずつ選んでテストケースをつくる

テスト技法　境界値分析法

同値クラスの境界値にバグが潜んでいることが多い

- 境界値と中間値 (同値クラス内でちょうど真ん中の値) をテストしてやるのが普通である。
- ただし、境界に上限や下限がない場合は一般的な値を用いる。

同値クラス	境界値	中間値 (または一般的な値)
~ -1	-1	-10
0 ~ 6	0 と 6	3
7 ~ 64	7 と 64	35
65 ~	65	75

分岐のあるプログラムのテスト
全ての分岐を網羅するテストを行うことが望ましい

```
public static int branch(int a, int b) {
  int sign;
  if (a == 0) {
    return 0; // (1)
  } else if (a > 0) {
    sign = 1; // (2)
  } else {
    sign = -1; // (3)
  }
  if (b == 0) {
    return a; // (4)
  } else {
    return sign * b; // (5)
  }
}
```

(1) を実行
(2) を実行した後に (4) を実行
(2) を実行した後に (5) を実行
(3) を実行した後に (4) を実行
(3) を実行した後に (5) を実行

最小限のテスト

```
branch(0, 0)
branch(1, 0)
branch(1, 1)
branch(-1, 0)
branch(-1, 1)
```

ループのあるプログラムのテスト
全ての分岐を網羅するテストを行うことが望ましい

- 繰り返し回数が0回になる (一度も繰り返さない) ようなパターン
- 繰り返し回数が1回になる (一度だけ実行する) ようなパターン
- よく使用されると考えられる回数で繰り返すようなパターン
- 想定した繰り返し回数の最大値になる (できる限り繰り返す) ようなパターン
- 想定した繰り返し回数の最大値 - 1 になる (できる限り繰り返す - 1) ようなパターン
- 想定した繰り返し回数の最大値 + 1 になる (できる限り繰り返す + 1) ようなパターン

```
public static int sum(int n) {
  int total = 0;
  for (int i = 1; i <= n; i++) {
    total += i;
  }
  return total;
}
```

このような場合、引数に 0, 1, 10, 100000 などの値を入れてやるとよい

一緒にやってみよう

- 今回の演習で使うテストドライバをいつものように指示通り正確にインストールする
  - テストドライバの導入に成功すると
    - プロジェクト「java20XX」の中の「test」というフォルダに「j1.lesson10.xml」という名前のファイルが作成される。
    - このファイルには今週使用するテスト一式が記述されている。
- j1.lesson10 というパッケージを作成する
- 演習資料にある擬似コード、テストの技法に関する例を手順通りに実行せよ。

課題1004のヒント

```
main
    m, n の値をプロンプト付で入力
    power を m, n を引数にして呼び出し、結果を適切に表示する

power(m, n)
    if n が 0
        1 をリターン
    if n が負
        power を m, -n (正)を引数にして呼び出し、結果の逆数をリターン
        // 1/m を -n 乗すると考えても同じ

    power を m, n/2 を引数にして呼び出し、結果を変数 sub に代入

    if n が偶数
        sub * sub をリターン
    else    //  n が奇数
        m * sub * sub をリターン
```