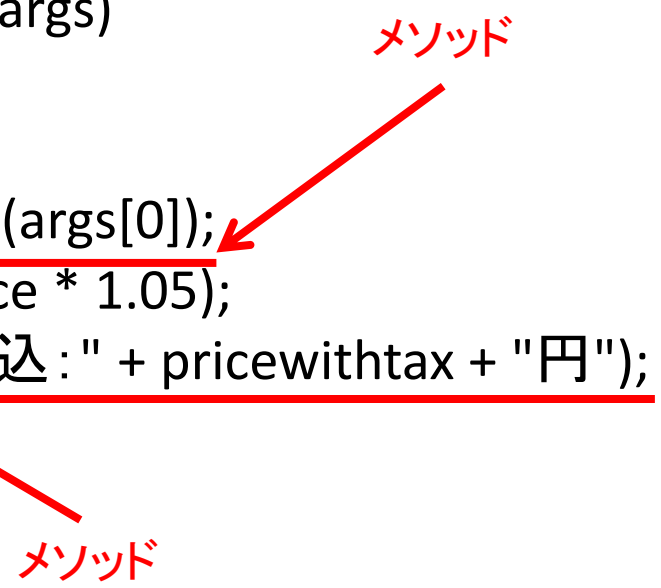
プログラミング基礎

第3回

メソッド

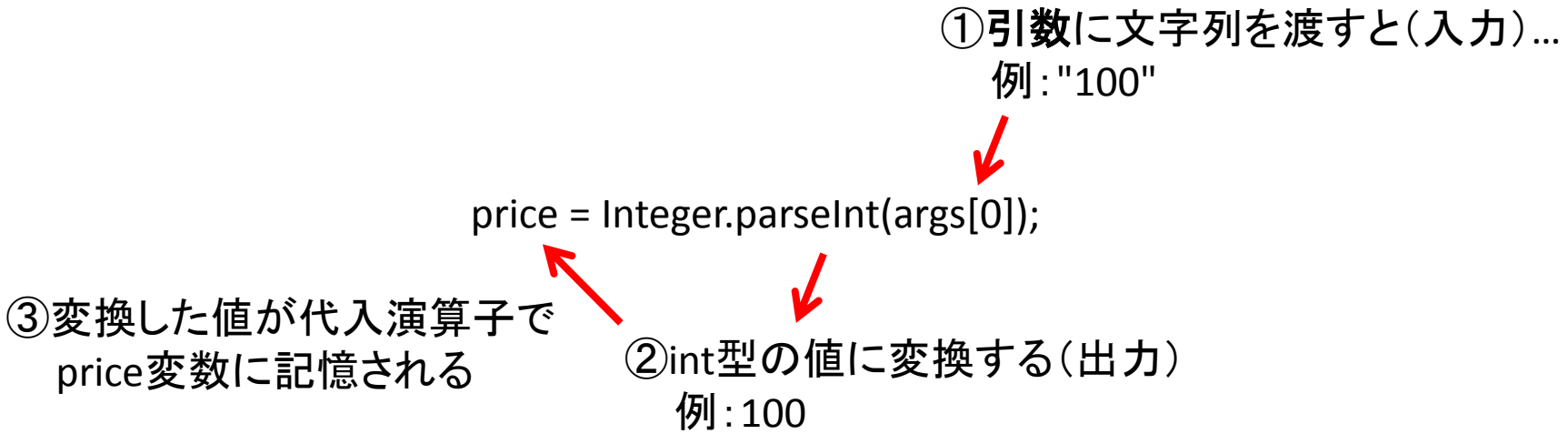
- 先日の「コマンドライン引数」に指定された値の，消費税込の値段を求める問題

```
public class Tax
{
        static public void main(String[] args)
        {
                int price, pricewithtax;
                price = Integer.parseInt(args[0]);
                pricewithtax = (int)(price * 1.05);
                System.out.println("税込：" + pricewithtax + "円");
        }
}
```



メソッド

- 特定の機能を実現するプログラムのかたまり



②がInteger.parseIntメソッドの機能，ということになる．
では，プログラムのかたまりはどこに？

メソッド

- 特定の機能を実現するプログラムのかたまり

Integer.parseInt →

JDK内にあるsrc.zip/java/lang/Integer.java内
（同ファイルより引用）

```
 *                                      representation to be parsed
 * @param     radix   the radix to be used while parsing <code>s</code>.
 * @return    the integer represented by the string argument in the
 *            specified radix.
 * @exception  NumberFormatException if the <code>String</code>
 *                                      does not contain a parsable <code>int</code>.
 */
public static int parseInt(String s, int radix)
                                      throws NumberFormatException
{
  if (s == null) {
    throw new NumberFormatException("null");
  }

                if (radix < Character.MIN_RADIX) {
                  throw new NumberFormatException("radix " + radix +

                  " less than Character.MIN_RADIX");
                }

                if (radix > Character.MAX_RADIX) {
                  throw new NumberFormatException("radix " + radix +
                  " greater than Character.MAX_RADIX");
                }

                int result = 0;
                boolean negative = false;
                int i = 0, max = s.length();
                int limit;
                int multmin;
                int digit;

                if (max > 0) {
                  if (s.charAt(0) == '-') {
                                      negative = true;
                                      limit = Integer.MIN_VALUE;
                                      i++;
                  } else {
                                      limit = -Integer.MAX_VALUE;
                  }
                  multmin = limit / radix;
                  if (i < max) {
                                      digit = Character.digit(s.charAt(i++),radix);
                                      if (digit < 0) {
                                        throw NumberFormatException.forInputString(s);
                                      } else {
                                        result = -digit;
                                      }
                  }
                  while (i < max) {
                                      // Accumulating negatively avoids surprises near MAX_VALUE
                                      digit = Character.digit(s.charAt(i++),radix);
                                      if (digit < 0) {
                                        throw NumberFormatException.forInputString(s);
                                      }
                                      if (result < multmin) {
                                        throw NumberFormatException.forInputString(s);
                                      }
                                      result *= radix;
                                      if (result < limit + digit) {
                                        throw NumberFormatException.forInputString(s);
                                      }
                                      result -= digit;
                  }
                } else {
                  throw NumberFormatException.forInputString(s);
                }
                if (negative) {
                  if (i > 1) {
                                      return result;
                  } else {            /* Only got "-" */
                                      throw NumberFormatException.forInputString(s);
                  }
                } else {
                  return -result;
                }
}

/**
 * Parses the string argument as a signed decimal integer. The
 * characters in the string must all be decimal digits, except that
 * the first character may be an ASCII minus sign <code>'-'</code>
 * (<code>'&amp;#92;u002D'</code>) to indicate a negative value. The resulting
 * integer value is returned, exactly as if the argument and the radix
```

メソッド

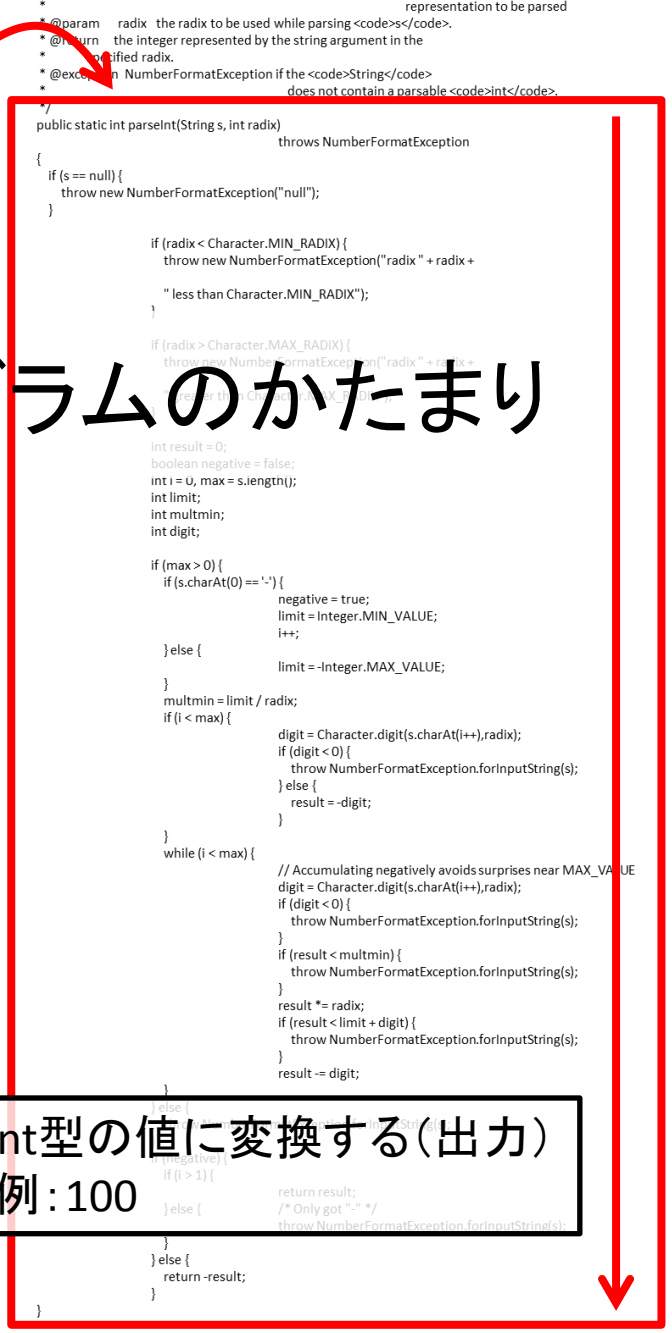
- 特定の機能を実現するプログラムのかたまり

①**引数**に文字列を渡すと（入力）…
例："100"

price = Integer.parseInt(args[0]);

③変換した値が代入演算子で
price変数に記憶される

②int型の値に変換する（出力）
例：100



```
                                                representation to be parsed
 * @param     radix   the radix to be used while parsing <code>s</code>.
 * @return    the integer represented by the string argument in the
 *            specified radix.
 * @exception NumberFormatException if the <code>String</code>
 *                                  does not contain a parsable <code>int</code>.
 */
public static int parseInt(String s, int radix)
                                  throws NumberFormatException
{
  if (s == null) {
    throw new NumberFormatException("null");
  }

                if (radix < Character.MIN_RADIX) {
                  throw new NumberFormatException("radix " + radix +

                  " less than Character.MIN_RADIX");
                }

                if (radix > Character.MAX_RADIX) {
                  throw new NumberFormatException("radix " + radix +
                  " greater than Character.MAX_RADIX");
                }

                int result = 0;
                boolean negative = false;
                int i = 0, max = s.length();
                int limit;
                int multmin;
                int digit;

                if (max > 0) {
                  if (s.charAt(0) == '-') {
                                  negative = true;
                                  limit = Integer.MIN_VALUE;
                                  i++;
                  } else {
                                  limit = -Integer.MAX_VALUE;
                  }
                  multmin = limit / radix;
                  if (i < max) {
                                  digit = Character.digit(s.charAt(i++),radix);
                                  if (digit < 0) {
                                    throw NumberFormatException.forInputString(s);
                                  } else {
                                    result = -digit;
                                  }
                  }
                  while (i < max) {
                                  // Accumulating negatively avoids surprises near MAX_VALUE
                                  digit = Character.digit(s.charAt(i++),radix);
                                  if (digit < 0) {
                                    throw NumberFormatException.forInputString(s);
                                  }
                                  if (result < multmin) {
                                    throw NumberFormatException.forInputString(s);
                                  }
                                  result *= radix;
                                  if (result < limit + digit) {
                                    throw NumberFormatException.forInputString(s);
                                  }
                                  result -= digit;
                  }
                } else {
                  throw NumberFormatException.forInputString(s);
                }
                if (negative) {
                  if (i > 1) {
                                  return result;
                  } else {        /* Only got "-" */
                                  throw NumberFormatException.forInputString(s);
                  }
                } else {
                  return -result;
                }
}

/**
 * Parses the string argument as a signed decimal integer. The
 * characters in the string must all be decimal digits, except that
 * the first character may be an ASCII minus sign <code>'-'</code>
 * (<code>'&amp;#92;u002D'</code>) to indicate a negative value. The resulting
 * integer value is returned, exactly as if the argument and the radix
```

メソッド

- 文字列をint型の数に変換する度に
  これ ⟶
  をmainメソッド内に記述するのは大変

- そこで，実際の処理内容を
  知らなくても簡単に使え，
  使いまわせるように，
  特定の機能を実現する
  プログラムのかたまりがメソッド

```
                                                    representation to be parsed
 * @param     radix   the radix to be used while parsing <code>s</code>.
 * @return    the integer represented by the string argument in the
 *            specified radix.
 * @exception  NumberFormatException if the <code>String</code>
 *                                   does not contain a parsable <code>int</code>.
 */
public static int parseInt(String s, int radix)
                                   throws NumberFormatException
{
  if (s == null) {
    throw new NumberFormatException("null");
  }

                if (radix < Character.MIN_RADIX) {
                  throw new NumberFormatException("radix " + radix +

                  " less than Character.MIN_RADIX");
                }

                if (radix > Character.MAX_RADIX) {
                  throw new NumberFormatException("radix " + radix +

                  " greater than Character.MAX_RADIX");
                }

                int result = 0;
                boolean negative = false;
                int i = 0, max = s.length();
                int limit;
                int multmin;
                int digit;

                if (max > 0) {
                  if (s.charAt(0) == '-') {
                                   negative = true;
                                   limit = Integer.MIN_VALUE;
                                   i++;
                  } else {
                                   limit = -Integer.MAX_VALUE;
                  }
                  multmin = limit / radix;
                  if (i < max) {
                                   digit = Character.digit(s.charAt(i++),radix);
                                   if (digit < 0) {
                                     throw NumberFormatException.forInputString(s);
                                   } else {
                                     result = -digit;
                                   }
                  }
                  while (i < max) {
                                   // Accumulating negatively avoids surprises near MAX_VALUE
                                   digit = Character.digit(s.charAt(i++),radix);
                                   if (digit < 0) {
                                     throw NumberFormatException.forInputString(s);
                                   }
                                   if (result < multmin) {
                                     throw NumberFormatException.forInputString(s);
                                   }
                                   result *= radix;
                                   if (result < limit + digit) {
                                     throw NumberFormatException.forInputString(s);
                                   }
                                   result -= digit;
                  }
                } else {
                  throw NumberFormatException.forInputString(s);
                }
                if (negative) {
                  if (i > 1) {
                                   return result;
                  } else {         /* Only got "-" */
                                   throw NumberFormatException.forInputString(s);
                  }
                } else {
                  return -result;
                }
}

/**
 * Parses the string argument as a signed decimal integer. The
 * characters in the string must all be decimal digits, except that
 * the first character may be an ASCII minus sign <code>'-'</code>
 * (<code>'&amp;#92;u002D'</code>) to indicate a negative value. The resulting
 * integer value is returned, exactly as if the argument and the radix
```

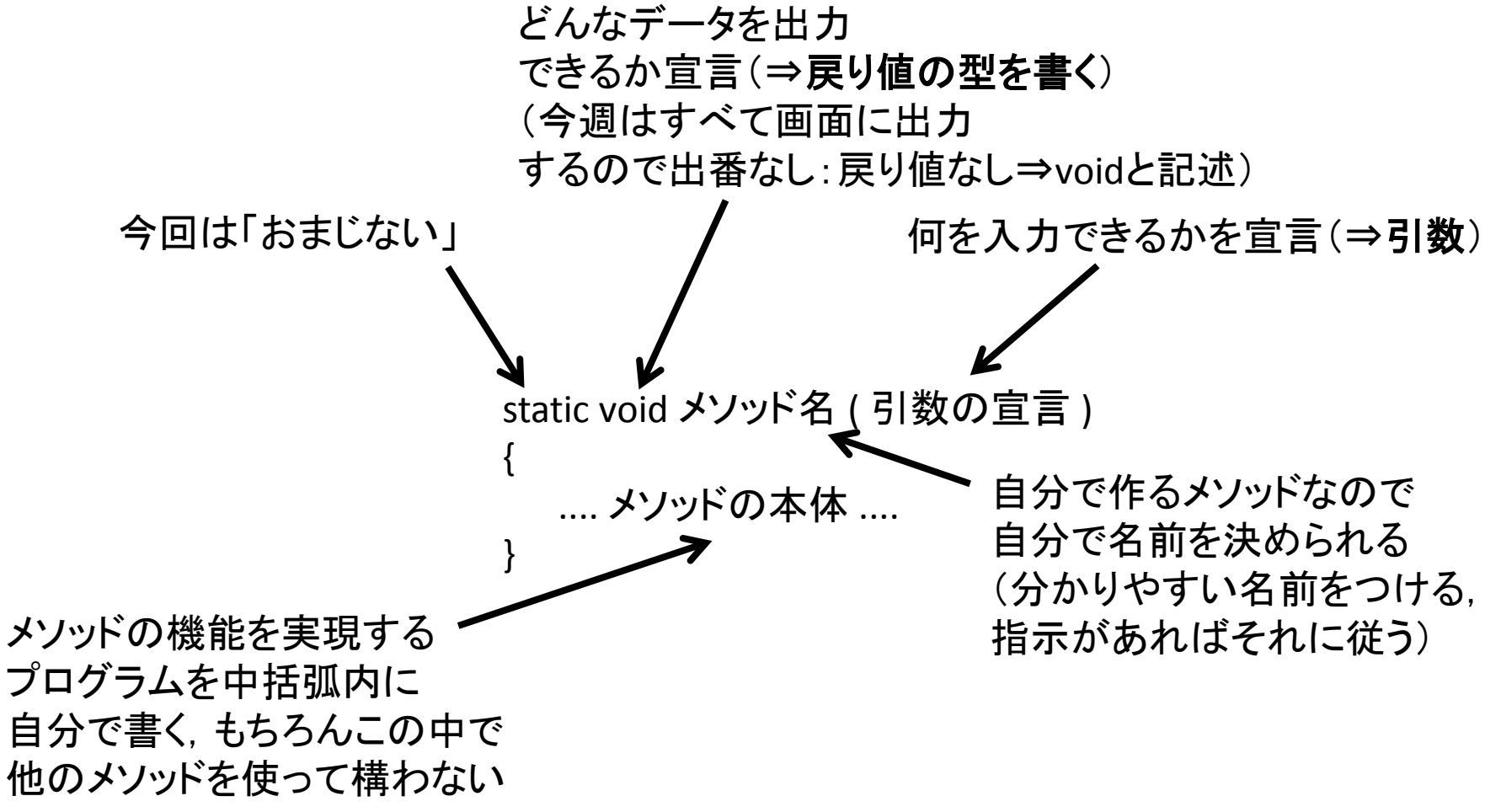
メソッド

- 必要であるが，用意されていない機能のメソッドは自分で作る必要がる

⇒今週，来週はメソッドの作り方を学ぶ

メソッドの書き方

- Web資料より引用



引数の宣言

- 具体例

```
public static int parseInt(String s)
{
    …
}
```

・parseIntメソッドはString型のデータを入力として受け取りますよ
・parseIntメソッドではそのデータを仮にsと名付けます⇒**仮引数**
＊変数と書き方が同じだが，意味が違うので注意

実際にこのメソッドを使うときはString型のデータを引数に指定する
実際に使われるデータ⇒**実引数**

使用例：

```
price = Integer.parseInt("1000");
price = Integer.parseInt(args[0]);
```

引数の宣言

- 入力は複数渡すこともできる
  ＝引数は複数宣言できる

```
static int mul(int x, int y)
{
    return x * y;
}
```

カンマで区切る

使用例：

```
int x = 100; int y;
y=(x, 10);
y=(1, 2);
```

作ったメソッドはどこに置く？

- 「public class プログラム」に続く中括弧の中に置く
  ⇒メソッドは常にクラスに属するモノである，ということ（mainメソッドも同様）

```
public class プログラム
{
        static public void main(String[] args)
        {
                …
        }

        static void 作ったメソッド(int  x)
        {
              ….
        }
}
```

作ったメソッドはどこに置く？

- 「public class プログラム」に続く中括弧の中に置く
  ⇒メソッドは常にクラスに属するモノである，ということ（mainメソッドも同様）

```
public class プログラム
{
        static void 作ったメソッド(int  x)
        {
            ....
        }

        static public void main(String[] args)
        {
            ...
        }
}
```

「public class プログラム」に続く中括弧の中であればどこでも構わない

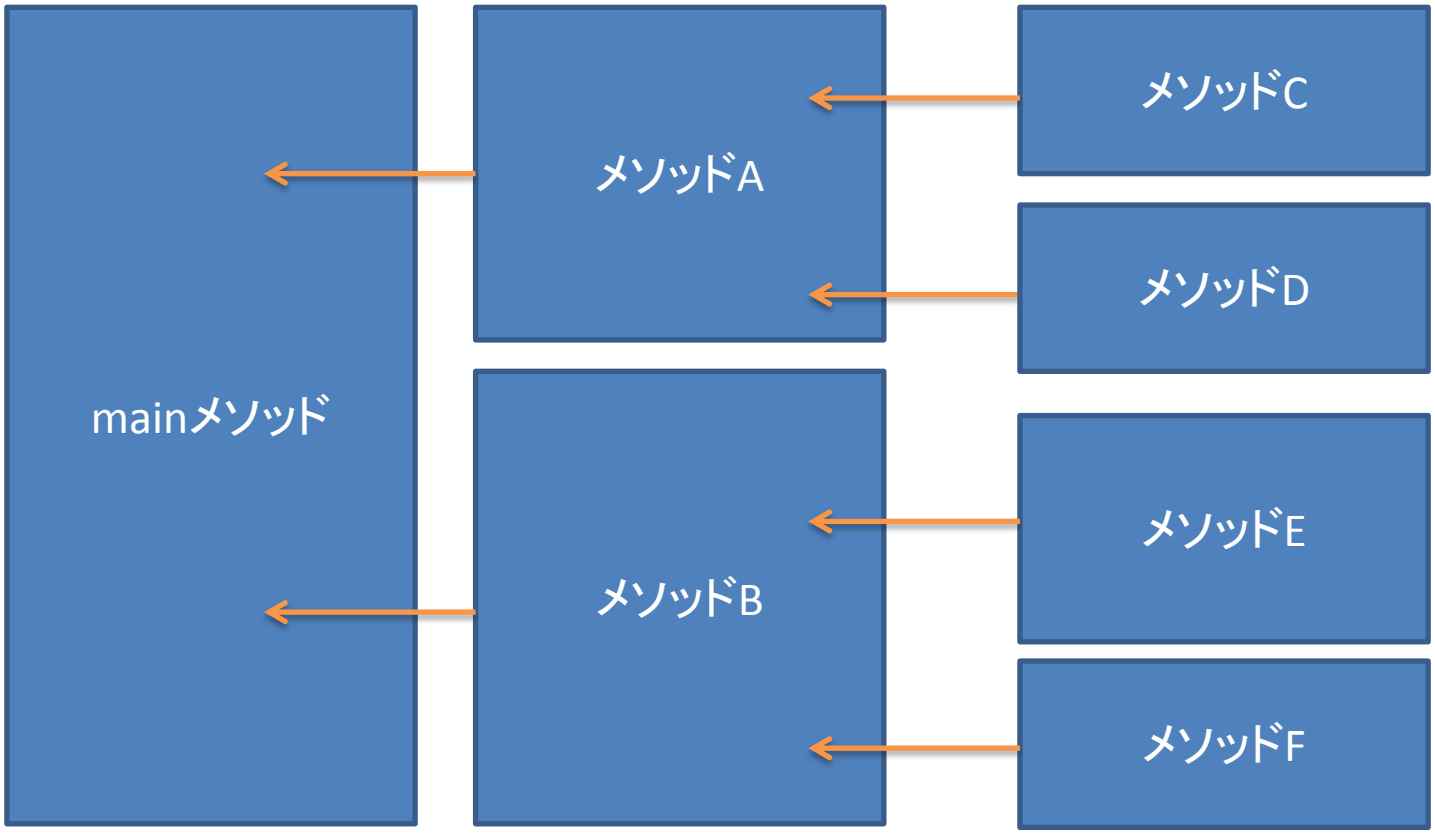
演習

- int型のデータが引数として渡された時，
  0では日
  1では月
  2では火
  3では水
  4では木
  5では金
  6では土と表示する
  intToDayOfTheWeekメソッドを作りなさい．
  また，mainメソッド内でテストすること．

  プログラム名はDayOfTheWeekとする．

メソッド

- mainメソッドは他のメソッドから成り立つ
  そのメソッドはさらに他のメソッドを使用する場合もある



もう少し一般的に

- 問題（作成するプログラム）は分離せよ
- 大きな問題を漠然と考えるのではなく，
  小さな問題に切り分け個々に取り組み，
  最後に組み合わせる
- おすすめ図書
  訳：柿内 賢信 著：G.ポリア
  「いかにして問題をとくか」
  丸善株式会社

メソッド

- 利点：作業を分散できる
  ⇒「ある機能を持つメソッドCはAさんが作ってください」
  ⇒「メソッドE，FはBさんが作ってください」

- 利点：テストが容易になる
  ⇒うまく動作しない箇所があれば，該当するメソッドのみでテストする

- 利点：再利用が出来る
  ⇒必要になった機能があれば，該当するメソッドのみ利用すればよい

⇒モジュールと呼ばれる考え方

実際現場では

- 作ったメソッドの単体テスト，結合テスト
  etc.
  ⇒作成したメソッドが仕様書通りに動くのか
  　チェックする作業

関連

- どうすれば効率的にソフトウェア開発ができるのか
  ⇒「ソフトウェア工学」と呼ばれるジャンル

スコープ

- 変数はその変数が属する中括弧内でのみ使える

```
public class ScopeSample
{
   public static void main(String[] args)
   {
      if (args.length == 2)
      {
         int x = Integer.parseInt(args[0]); int y = Integer.parseInt(args[1]);
         System.out.println(mul(x, y));
      }
      else
      {
         System.out.println("引数を2つ指定してください");
      }
   }

   static int mul(int x, int y)
   {
      int r = x * y;
      return r;
   }
}
```

変数x，yこの中括弧内だけで使える

変数rこの中括弧内だけで使える

演習

- 今日の日付のページに掲載されているshowScoresメソッドのエラー・バグを取りなさい
  showScoresメソッド:
  1名以上の点数を記憶しているint型の配列を引数として渡すと，最低点，最高点，平均点を表示する

実のところ

- mainメソッドもれっきとしたメソッドの一種

（入力できるデータが文字列配列に限定されているが）
（コマンドライン）引数を受け取って（＝入力）

```
public class Tax
{
        static public void main(String[] args)
        {
            int price, pricewithtax;
            price = Integer.parseInt(args[0]);
            pricewithtax = (int)(price * 1.05);
            System.out.println("税込:" + pricewithtax + "円");
        }
}
```

値段を引数として消費税込の値段を表示するプログラムのかたまり

消費税込の値段を表示する
⇒このmainメソッドの機能

実のところ

- ユーザから見れば作られたプログラムはメソッドの一種