

06 関数



Created by GT F

Last updated just a moment ago

- [関数の基本構成](#)
- [関数の戻り値とReturn](#)
- [関数の使用（関数の使用手順とIPO）](#)
- [関数のオーバーロード](#)
- [可変長引数](#)
- [再帰](#)
- [専門用語](#)
- [質問](#)
- [補足](#)
 - [なぜ関数を定義する（汎用化の説明）](#)
 - [メソッド同士は兄弟関係](#)
 - [戻り値あり&戻り値ない](#)

汎用（はんよう）的に使用するコードは、関数という1つの単位にまとめると、繰り返し使用できるようになります。現章でも、既にmainという関数を取り上げました。

i main関数はプログラムの先頭で呼び出せる特別の関数である。（main関数は汎用化の目的ではありません。）

関数の基本構成

関数は4要素がもっています。[修飾子](#)（複数可能性があります）、[返却値の型](#)、[関数名](#)、[引数](#)（複数可能性があります）。

```
1 修飾子 返却値の型 関数名 ( 引数1の型 引数1, 引数2の型 引数2 ) {  
2    //... 処理内容  
3 }
```

1. 関数の一つ特徴は `()` であること。
2. `()` の前は必ず [関数名](#) であること。
3. [関数名](#) の前は必ず [返却値型（戻り値型）](#) であること。
4. `()` の中に必ず [引数](#) であること。

例

```
1 public static void main(String[] args) {  
2     System.out.println(args[0]);  
3 }
```

- 修飾子： `public` (公開) と `static` (静的) ※本書は理解しなくてもよい
- 返却値の型は `void` の為、返却値なし
- 関数名は `main`
- 引数が1つあります。引数名は `args`, 引数の型は `String[]` (Stringの配列型)
- 戻り値がある場合、メソッド処理に必ず `return` ステートメントがあること。

質問：以下関数（メソッド）の返却型は？

```
1 public String getName() {
2     return "name";
3 }
```

i 関数定義ロジックにさらに関数定義できません。

新人さんよく間違いところ。

```
1 public void method() {
2     public void method2() {
3         // NG 関数内部関数を再定義できない。
4     }
5 }
```

関数の主体 `{}`、ロジックのみ許容、定義など行えません。

関数の戻り値とReturn

関数の戻り値は `void` ではない場合、`return` のステートメントで値を返却（へんきゃく）しなければいけません。`return` キーワードを実施後、メソッドが終了させます。`return` 以後のコードが到着できなくなります。

質問：以下関数は不正です。（デッドコード）

```
1 public String method() {
2     return ""; // メソッドを既に終了
3     int a = 10; // dead code
4 }
5 public String method() {
6     while(1==1) {}
7     int a = 10; // 無限ループの原因で dead code
8 }
```

返却値はStringの関数

```
1 public String functionReturnString() {return "";}

```

返却値がない関数

```
1 public void noReturnFunction() {}

```

引数がない関数

```
1 public void noParamter() {}
```

引数が複数あり関数

```
1 public void manyParamter(int a, float b, double c) {}
```

実例：重力加速度 $y = g \cdot t^2 / 2$ は以下関数で定義する。

1. 引数 t , 型は float 型
2. 関数名は gAcceleration（先頭文字は半角英字設定必要があり）
3. 返却値は float 型
4. 関数の処理ロジック： $g \cdot t^2 / 2$ ($g=9.8F$)

上記手順通り、以下メソッドを定義。

```
1 // メソッド定義
2 public static float gAcceleration(float t) {
3     float y = 9.8F * t * t / 2;
4     return y; // 変数y値の返却する
5 }
```

※return リターン。右辺の式 $9.8F * t * t / 2$ の値を返却する意味。public, static 本書は理解しなくてもいいです。

関数の使用（関数の使用手順とIPO）

関数の使用（関数の呼び出す）方法は関数名でメソッドを呼び出す。必要な引数（パラメータ）を引き渡す。返却値がある場合、返却値を受け取り。

上記メソッド「重力加速度」を呼び出す手順は

1. 呼び出すメソッド名 gAcceleration。⇒関数名は？
2. 引数のパターン t に値を設定して引き渡す。⇒なにが必要ですか
3. 返却値は float の為、float変数を宣言して、値を代入する。⇒なにを戻しますか


```
1 public static void main(String[] args) {
2     float t = 1.0F; // 引数として関数に渡す
3     float y = gAcceleration(t); // 関数の戻り値を変数 y に代入する
4     System.out.println(y);
5 }
```

IPOとは。I = Input, P = process, O = output

- Input インプット：関数の引数など
- Process プロセス：関数の処理ロジック（呼び出す元は意識しなくてもいいです）
- Output アウトプット：関数の戻り値

質問：以下メソッドのIPOで分析してください。


```
1 String name = "abc";
2 int length = name.length(); // I=なし 0=3
3 int index = name.indexOf('b'); // I='b' 0=1
```

 Eclipse & Netbeansのショートカット：メソッドへ飛び出す：メソッド名を **CTRL** 押しなが
ら、左クリックする。

関数のオーバーロード

引数の**数**又は引数の**型**は異なる又は引数型の**順**場合、メソッド名は重複可能。JDKは自動的にINPUTを
対して、メソッドを選びます。

```
1 public void sampleMethod() {}
2 public void sampleMethod(int a) {} // 引数
3 public void sampleMethod(int a, int b) {} //
4 public void sampleMethod(int c) {} // NG 不可、メソッド#2と重複
5 public void sampleMethod(float a) {} // OK 引数の型が異なる
6 public int sampleMethod(char c) { return 1; } // NG不可、メソッド#1到着
```

 Javaプログラミングは必ず「型」を意識してください。名称（変数名、クラス名等）ではな
い。

可変長引数

Javaのメソッドの**最後の引数**は可変長引数として使うこともできる。可変長変数は配列として利用
可能します。

```
1 public static void sample1(String[] args) {
2     System.out.println(args.length);
3 }
4 ↑↓同じ、下記「可変長引数」
5 public static void sample2(String...args) {
6     System.out.println(args.length);
7 }
```

正し、可変長変数メソッドの呼び出す方法がちょっと違います。

```
1 sample1(new String[]{"1", "2", "3"}); // 配列引数
2 sample2(); // OK 可変長引数長さ=0
3 sample2("1"); // OK 可変長引数長さ=1
4 sample2("1", "2"); // OK 可変長引数長さ=2
```

再帰

メソッドを自分から自分呼び出すのは再帰と呼びます。ここでは、階乗計算を再帰呼び出しにより実装する例を紹介する。

```
1 int fact(int n) {
2     if (n == 0) return 1; /* 脱出条件。0! は 1 である */
3     else return fact(n - 1) * n; /* n! は (n-1)! に n を乗じたもの。再帰呼出し */
4 }
```

※再帰メソッドの利用する場合、優先考えなければならないことは再帰の脱出条件。

専門用語

1. 修飾子（しゅうしょくし）
2. 関数（かんすう）
3. 呼び出す
4. 再帰

質問

質問 1．以下関数の修飾子、返却型、関数名、引数（引数の型・名称）を回答してください。

```
1 public static void main(String[] args);
```

質問 2．配列の和、MAX値、MIN値を求める処理を関数化してください、main関数に呼び出してください。

```
1 // 例：配列のMax値を求めるメソッド
2 public static int max(int[] datas) {
3     // ...処理を記載
4 }
```

質問 3．以下メソッドの実施結果を教えてください。

```
1 public class Kicker {
2     private static void sayHello(String y) {
3         y = "Hello world";
4     }
5     public static void main(String args[]) {
6         String x = null;
7         sayHello(x);
8         System.out.println(x);
9     }
10 }
```

質問 4：以下メソッドの実施結果を教えてください。

```
1 public class Kicker {
2     private static void fillArray(String[] array) {
3         array[0] = "value2";
4     }
5 }
```

```

5     public static void main(String args[]) {
6         String[] array = new String[]{"value1"};
7         fillArray(array);
8         System.out.println(array[0]);
9     }
10 }

```

質問 5：以下メソッドの実施結果を教えてください。

```

1  public class Kicker {
2      private static void fillArray(String[] array) {
3          array = new String[]{"1", "2"};
4      }
5      public static void main(String args[]) {
6          String[] array = null;
7          fillArray(array);
8          System.out.println(array == null);
9      }
10 }

```

質問 6 - 0：配列は直接に印刷できません。下記メソッドを利用して、配列をprintします。以下メソッドを呼び出してください。

```

1  public static void printArray(Object[] objs) {
2      if(objs == null) {
3          System.out.println("null");
4          return;
5      }
6      String prefix = "";
7      StringBuilder sb = new StringBuilder();
8      sb.append("[");
9      for(Object obj : objs) {
10         sb.append(prefix);
11         sb.append(String.valueOf(obj));
12         prefix = ", ";
13     }
14     sb.append("]");
15     System.out.println(sb.toString());
16 }

```

質問 6 - 1：以下静的なメソッドを定義してください。

1. メソッド名: megerArray
2. 引数 1：intの配列 left
3. 引数 2：intの配列 right
4. 戻り値：intの配列
5. 処理内容：引数left と 引数right をマージして、新しい配列を作成して返却する。

以下メソッドの実施例：

```

1  int[] left = new int[] {1, 2, 3, 4};
2  int[] right = new int[] {5, 6, 7, 8};

```

```
3 // メソッドを実施後
4 int[] result = new int[] {1, 2, 3, 4, 5, 6, 7, 8};
```

質問 6 - 2 : 以下静的なメソッドを定義してください。

1. メソッド名: subArray
2. 引数 1 : intの配列 array
3. 引数 2 : 開始のstartIndex
4. 引数 3 : 長さlength
5. 戻り値 : intの配列
6. 処理内容 : startIndexからstartIndex + lengthまでサブ配列を取得する

以下メソッドの実施例 :

```
1 int[] array = new int[] {1, 2, 3, 4};
2 // start = 1, length = 2
3 int[] result = new int[] {2, 3}
```

質問 7 : 以下静的なメソッドを定義してください。

1. メソッド名 : trim
2. 引数 1 : charの配列
3. 戻り値 : charの配列
4. 処理内容 : 引数 1 のcharの配列先頭と末尾の空白文字列を削除してください。

以下メソッドの実施例 :

```
1 char[] input = new char[] {' ', ' ', 'A', 'b', ' ', ' ', 'C', ' '}; // 入力引数
2 // メソッドを実施後
3 char[] output = new char[] {'A', 'b', ' ', ' ', 'C'}; // 入力引数
```

ヒント :

1. 配列の起点から、' 'ではない文字列のインデックスを探す。
2. 配列の終点から、' 'ではない文字列のインデックスを探す。
3. 起点と終点を引数として、質問 6 - 2 で作成したサブ配列メソッドを呼び出して、結果を求める。

質問 8 : バブルソート用メソッドを作成してください。

1. メソッド名 : bubbleSort
2. 引数 1 : intの配列
3. 戻り値 : ソート済みの配列
4. 処理内容 : bubbleSortアルゴリズムを用い、入力した引数の配列をソートしてください。

質問 9 : メソッドの呼び出す練習。

1. 質問 6 - 1 を用い、2 つ配列をマージします。
2. 質問 8 を用い、ステップ 1 のマージした配列をソートする。
3. 注意 : コードを 1 行にしてください。

補足

なぜ関数を定義する（汎用化の説明）

以下 2 配列にすべて要素の和を求め場合

```
1 public static void main(String[] args) {
2     int[] array1 = new int[]{1, 2, 3};
3     int sum1 = 0;
4     for(int i = 0; i < array1.length; ++i) {
5         sum1 = sum1 + array1[i];
6     }
7     System.out.println(sum1);
8
9     int[] array2 = new int[]{5, 6, 7};
10    int sum2 = 0;
11    for(int i = 0; i < array2.length; ++i) {
12        sum2 = sum2 + array2[i];
13    }
14    System.out.println(sum2);
15 }
```

上記サンプルコード、配列の和の求める処理は類似です。異なる箇所は `array1` と `array2` です。類似処理を汎用化する為、メソッドを定義する

メソッド同士は兄弟関係



戻り値あり&戻り値ない

戻り値あるメソッドはATMからお金を引き出す：金額を入力して、お金を出す。戻り値ないメソッドはATMに保存する：お金を入力、返却しない。

```
1 public int getMoneyFromATM(int amount) {  
2     return amount;  
3 }  
4  
5 public void saveMoneyToATM(int amount) {  
6     // 返却値ない  
7 }
```

 Like Be the first to like this

No labels 