Java1 **補講**

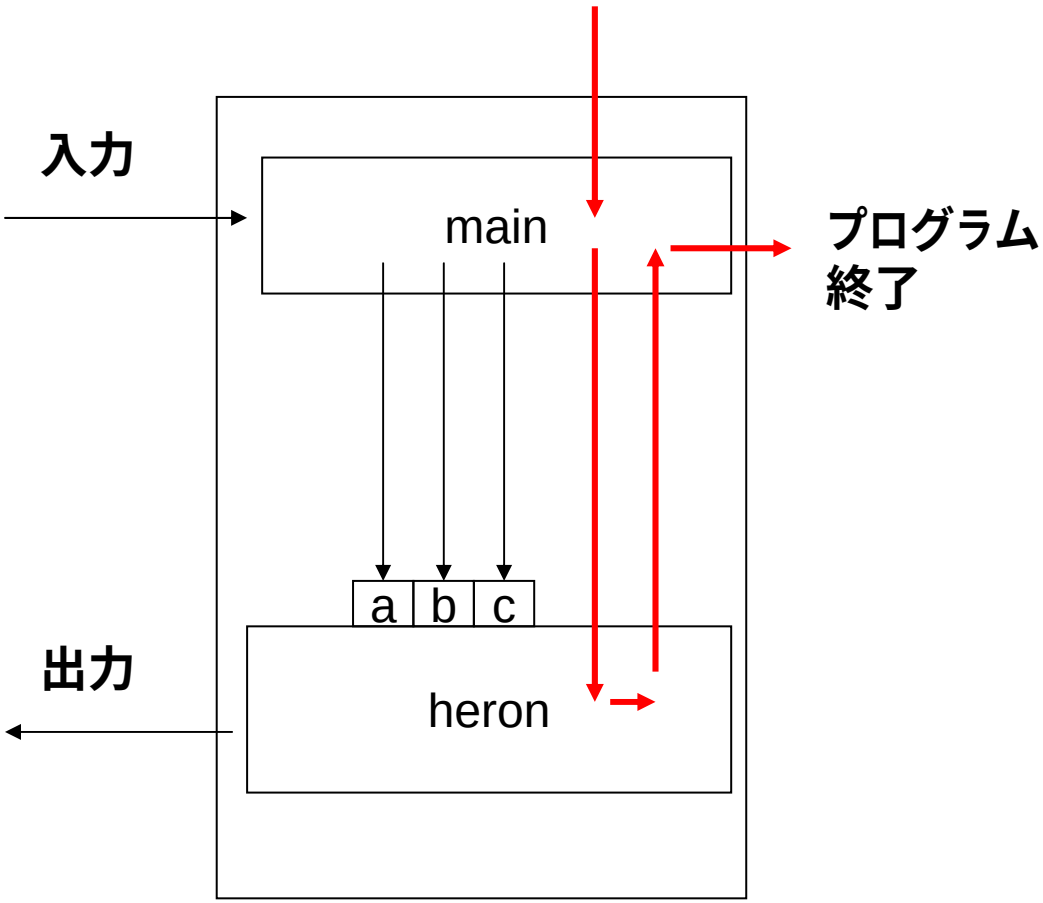
2007/7/19

戻り値の無いメソッド

HeronMethod

```
package j1.lesson07;
import java.io.*;

public class HeronMethod {

    public static void main(String[] args) throws IOException {
        BufferedReader reader =
            new BufferedReader(new InputStreamReader(System.in));
        System.out.print("辺aの長さを入力:");
        double a = Double.parseDouble(reader.readLine());
        System.out.print("辺bの長さを入力:");
        double b = Double.parseDouble(reader.readLine());
        System.out.print("辺cの長さを入力:");
        double c = Double.parseDouble(reader.readLine());

        heron(a, b, c);
    }

    public static void heron(double a, double b, double c) {
        double s = (a + b + c) / 2;
        double x = Math.sqrt(s * (s - a) * (s - b) * (s - c));
        System.out.println("三角形の面積は" + x);
    }
}
```

実行権とデータの流れ　責任の分担



main	heron
入力	**面積計算**
	面積の出力

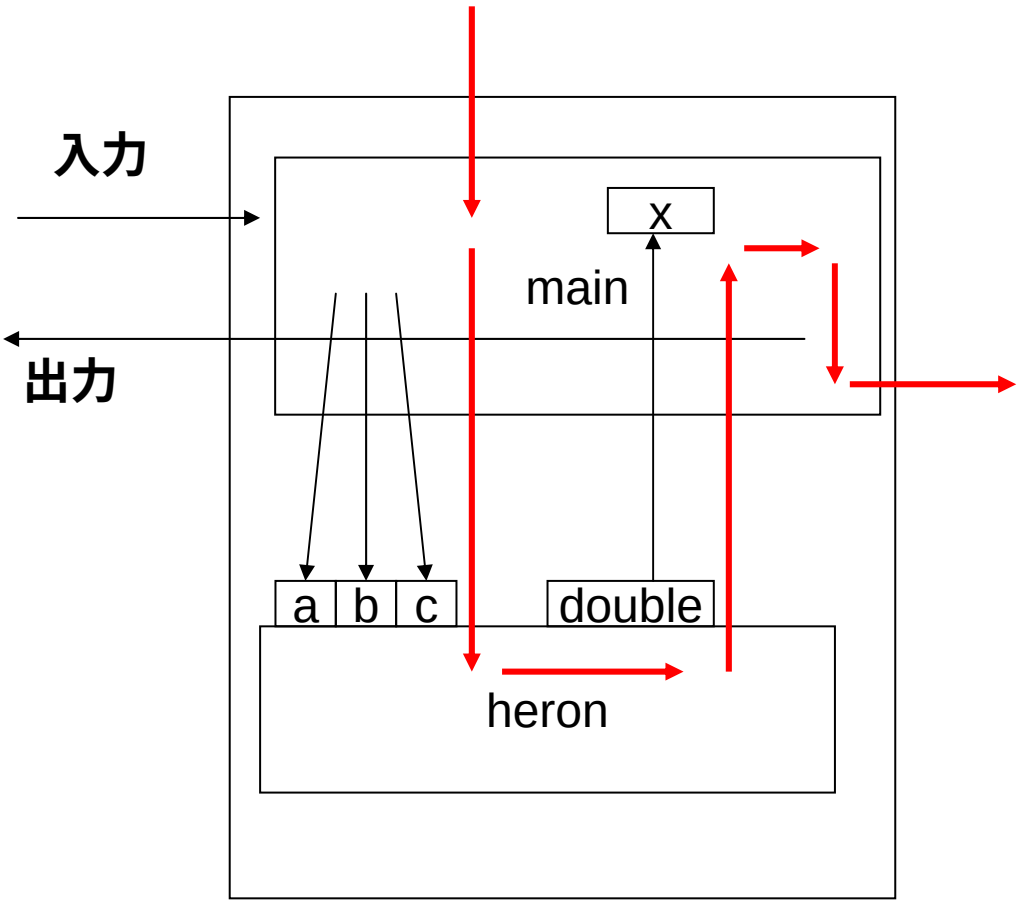
```
public static void heron(double a, double b, double c)
```

戻り値のあるメソッド

HeronMethod2

```
package j1.makeup;
import java.io.*;

public class HeronMethod2 {
    public static void main(String[] args) throws IOException {
        BufferedReader reader =
            new BufferedReader(new InputStreamReader(System.in));
        System.out.print("辺aの長さを入力:");
        double a = Double.parseDouble(reader.readLine());
        System.out.print("辺bの長さを入力:");
        double b = Double.parseDouble(reader.readLine());
        System.out.print("辺cの長さを入力:");
        double c = Double.parseDouble(reader.readLine());

        double x = heron(a, b, c);
        System.out.println("三角形の面積は" + x);
    }

    public static double heron(double a, double b, double c) {
        double s = (a + b + c) / 2;
        double x = Math.sqrt(s * (s - a) * (s - b) * (s - c));
        return x;
    }
}
```

実行権とデータの流れ　責任の分担



main	heron
入力	**面積計算**
面積の出力	

public static double heron(double a, double b, double c)

データのやり取り

- メソッド間でのデータのやり取りは主に以下の仕組みで行われる。
  - メソッドの引数
  - メソッドの戻り値
- 外界（コンソール、画面など）とのデータのやり取り
  - 入力（キーボード、マウス）
  - 出力（コンソール画面、グラフィックス）

実行の主体と実行権の移動

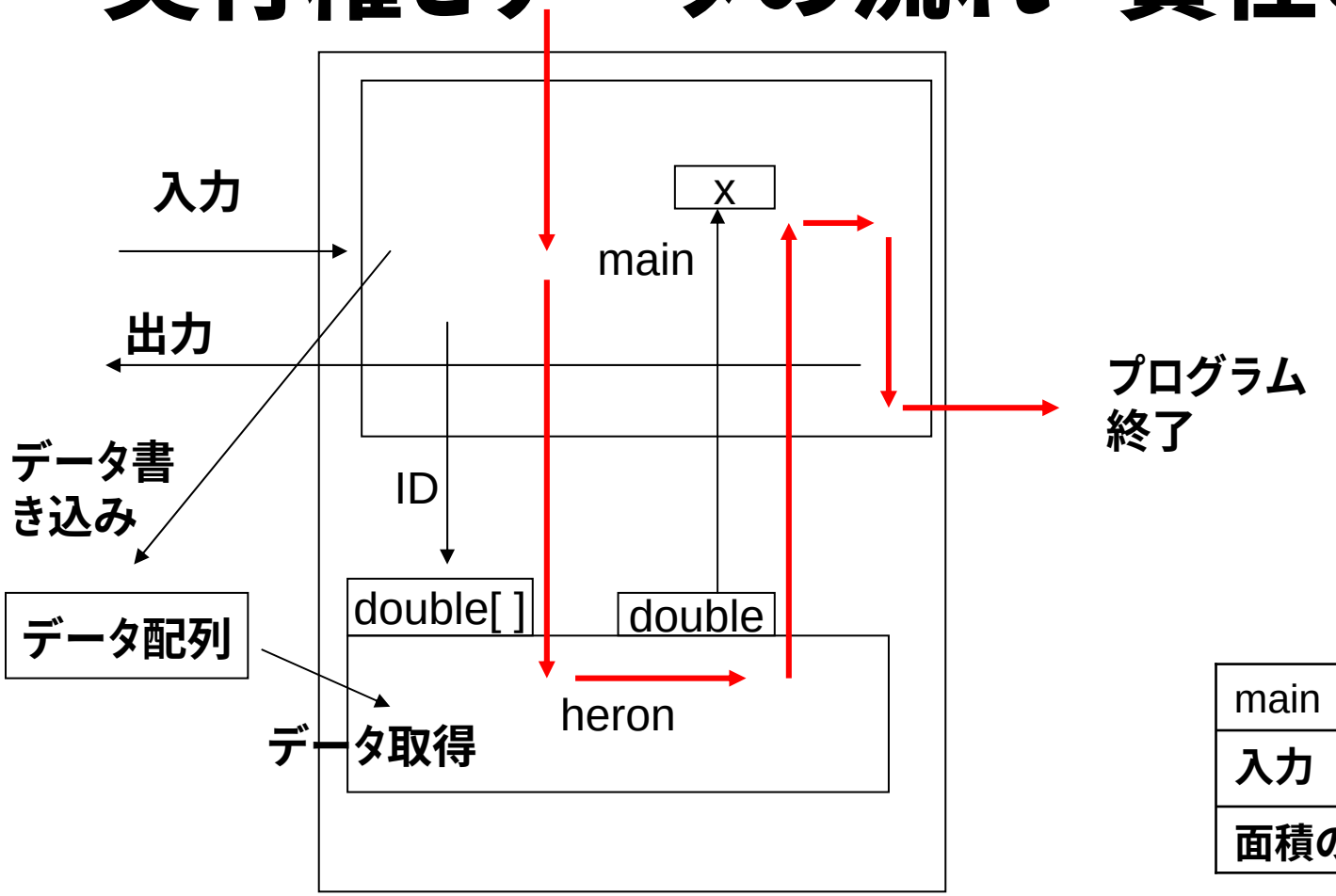
- プログラムを実行する主体はスレッド(thread)と呼ばれる。
- スレッドはメソッドを渡り歩く。
- スレッドはmainメソッドの実行から始める
  - プログラムの実行はmainメソッドの最初の命令から始まる
- メソッド呼び出しの命令を実行すると、スレッドは呼び出されたメソッドに移る。呼び出されたメソッドが実行を終了すると（return命令などによって）、スレッドは呼び出し側に戻る。

引数に配列をとるメソッド

HeronMethod3

```
package j1.makeup;
import java.io.*;

public class HeronMethod3 {
    public static void main(String[] args) throws IOException {
        BufferedReader reader =
            new BufferedReader(new InputStreamReader(System.in));
        double[] data = new double[3];
        System.out.print("辺aの長さを入力:");
        data[0] = Double.parseDouble(reader.readLine());
        System.out.print("辺bの長さを入力:");
        data[1] = Double.parseDouble(reader.readLine());
        System.out.print("辺cの長さを入力:");
        data[2] = Double.parseDouble(reader.readLine());

        double x = heron(data);
        System.out.println("三角形の面積は" + x);
    }

    public static double heron(double[] edges) {
        double s = (edges[0] + edges[1] + edges[2]) / 2;
        double x = Math.sqrt(s*(s-edges[0])*(s-edges[1])*(s-edges[2]));
        return x;
    }
}
```

実行権とデータの流れ　責任の分担



main	heron
入力	**面積計算**
面積の出力	

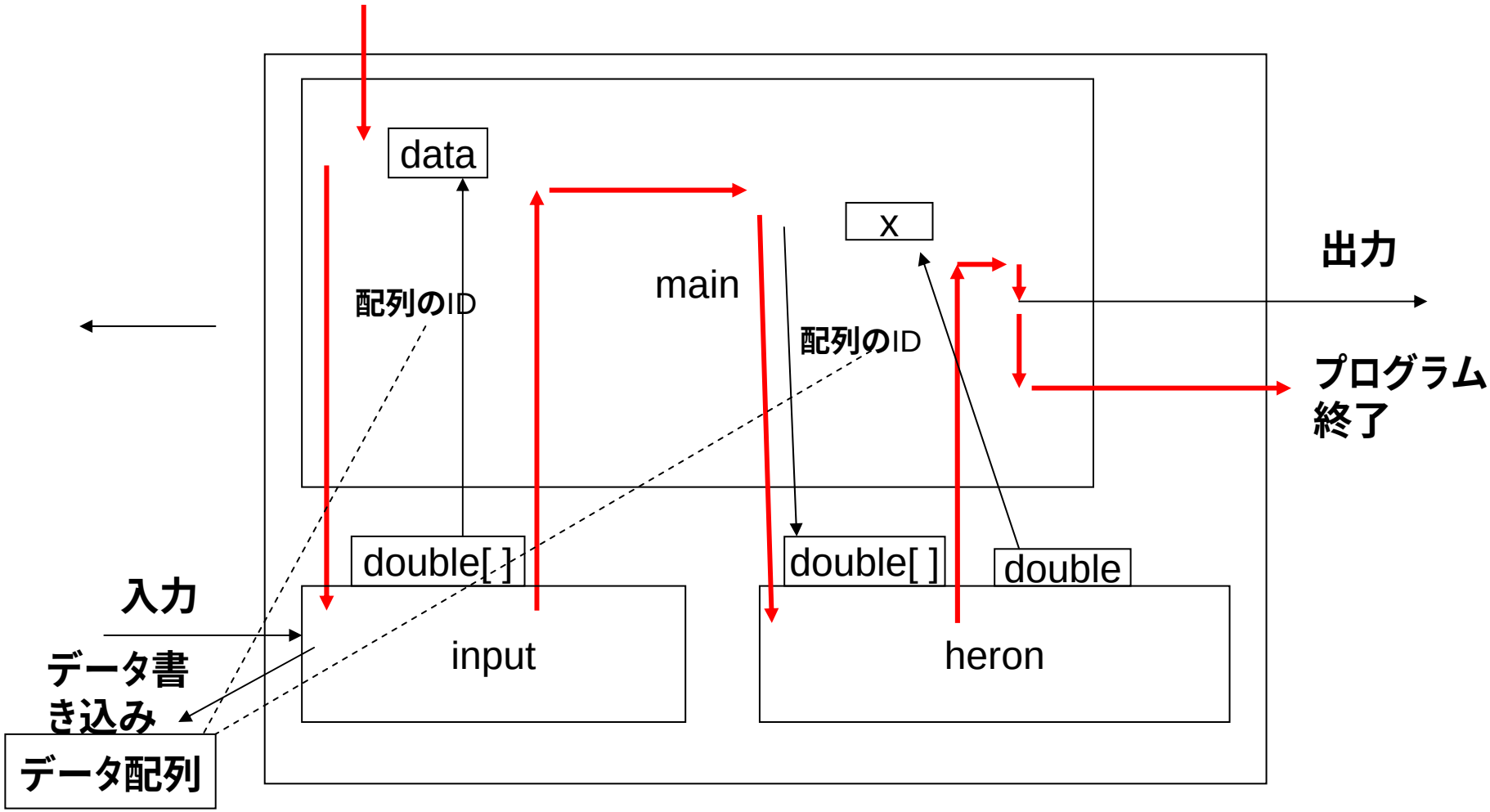
public static double heron(double[ ] edges)

戻り値に配列をとるメソッド

HeronMethod4

```
public class HeronMethod4 {
    public static void main(String[] args) throws IOException {
        double[] data = inputEdges();
        double x = heron(data);
        System.out.println("三角形の面積は" + x);
    }

    public static double[] inputEdges()throws IOException{
        BufferedReader reader =
            new BufferedReader(new InputStreamReader(System.in));
        double[] input = new double[3];
        System.out.print("辺aの長さを入力:");
        input[0] = Double.parseDouble(reader.readLine());
        System.out.print("辺bの長さを入力:");
        input[1] = Double.parseDouble(reader.readLine());
        System.out.print("辺cの長さを入力:");
        input[2] = Double.parseDouble(reader.readLine());
        return input;
    }

    public static double heron(double[] edges) {
        double s = (edges[0] + edges[1] + edges[2]) / 2;
        double x = Math.sqrt(s*(s-edges[0])*(s-edges[1])*(s-edges[2]));
        return x;
    }
}
```

実行権とデータの流れ



public static double[ ] inputEdges( )

public static double heron(double[ ] edges)

```
public class Distribution {
  public static void main(String[] args) throws IOException {
    BufferedReader reader =
      new BufferedReader(new InputStreamReader(System.in));
    System.out.print("受験者数を入力: ");
    int count = Integer.parseInt(reader.readLine());

    int[] scoreList = new int[count];
    for (int i = 0; i < count; i++) {
      System.out.print("点数を入力: ");
      scoreList[i] = Integer.parseInt(reader.readLine());
    }
    int[] distribution = createDistribution(scoreList);
    printDistribution(distribution);
  }

  public static int[] createDistribution(int[] scores) {
    int[] dist = new int[11];
    for (int i = 0; i < scores.length; i++)
      dist[scores[i] / 10]++;
    return dist;
  }

  public static void printDistribution(int[] dist) {
    for (int i = 0; i <= 9; i++)
      System.out.println(i * 10 + "-" + (i * 10 + 9) + ":" + dist[i]);
    System.out.println("100:" + dist[10]);
  }
}
```

得点のint型配列を引数にとり、得点分布を長さ11のint型配列で返す

```
public static int[] createDistribution(int[] scores) {
  int[] dist = new int[11];
  for (int i = 0; i < scores.length; i++)
    dist[scores[i] / 10]++;
  return dist;
}
```

得点分布のint型配列を引数にとり、表にして表示する

```
public static void printDistribution(int[] dist) {
  for (int i = 0; i <= 9; i++)
    System.out.println(i*10+"-"+(i*10+9)+":"+ dist[i]);
  System.out.println("100:" + dist[10]);
}
```

再帰呼び出し

講義スライドから

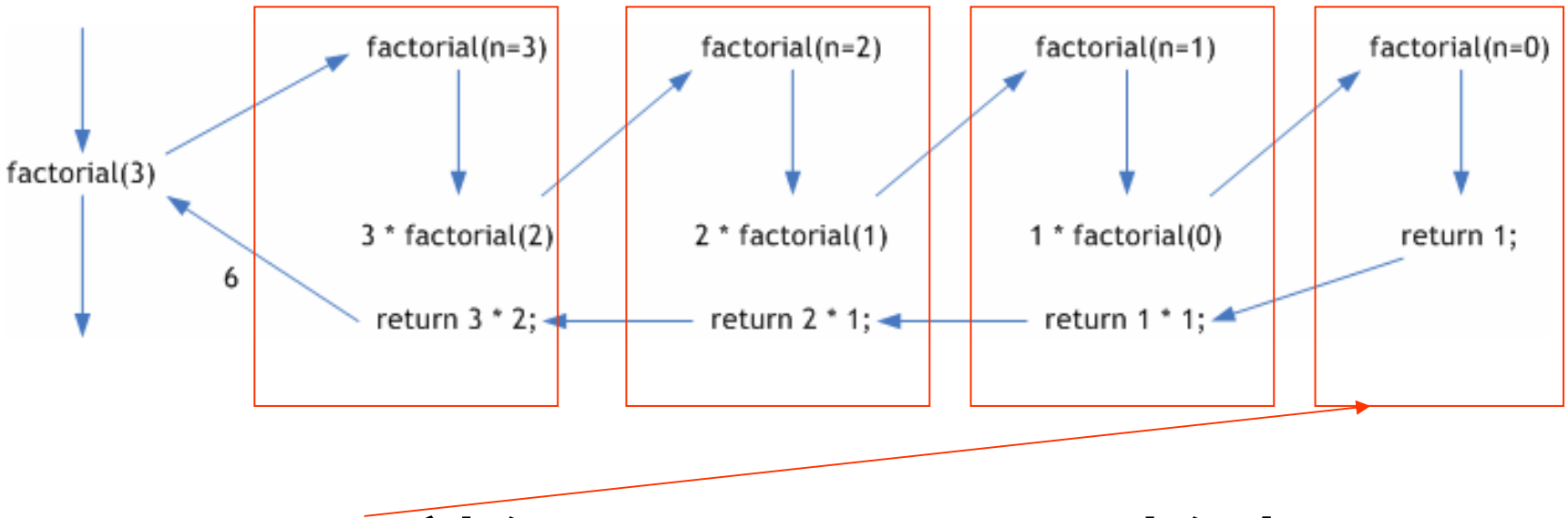
階乗を再帰的(recursive)に定義する

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(3) として起動すると



`factorial(0)`が実行されているときは4つの実行中あるいは再帰呼び出しからの帰り待ちの状態のメソッド起動がある。

`factorial`メソッドの仮引数である自動変数`n`は`factorial`の呼び出しの度に独立して自動的に作られ、消されるので、ソースプログラム上で同じ変数`n`であっても実行中の異なるメソッド起動では実体が異なる。

Fibonacciの解説

```
public class Fibonacci {
  static int count;

  public static void main(String[] args) {
    for (int i = 0; i <= 10; i++) {
      count = 0;
      System.out.println("fibonacci(" + i + ") = " + fibonacci(i));
      System.out.println("起動回数は" + count + "回");
    }
  }
  // フィボナッチ数列の 第 k 項を計算するメソッド
  public static int fibonacci(int k) {
    count++;
    if (k == 0) return 0;
    else if (k == 1) return 1;
    else
      return fibonacci(k - 1) + fibonacci(k - 2);
    }
  }
}
```

fibonacci(i)を起動したときに再帰的に呼び出された回数を数えるためのクラスフィールド

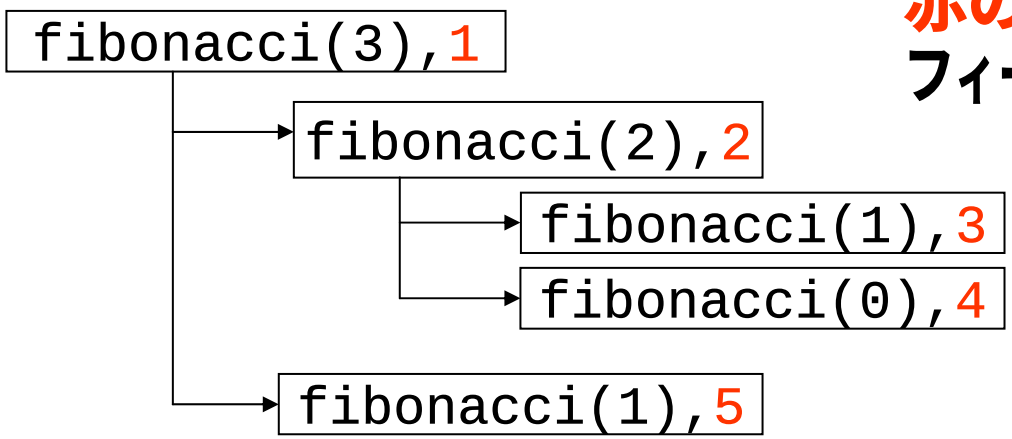
```
public class Fibonacci {
  static int count;

  public static void main(String[] args) {
    for (int i = 0; i <= 10; i++) {
      count = 0;
      System.out.println("fibonacci(" + i + ") = " + fibonacci(i));
      System.out.println("起動回数は" + count + "回");
    }
  }
  // フィボナッチ数列の 第 k 項を計算するメソッド
  public static int fibonacci(int k) {
    count++;
    if (k == 0) return 0;
    else if (k == 1) return 1;
    else
      return fibonacci(k - 1) + fibonacci(k - 2);
    }
  }
}
```

クラスフィールドの値を変更

```
public class Fibonacci {
  static int count;

  public static void main(String[] args) {
    for (int i = 0; i <= 10; i++) {
      count = 0;
      System.out.println("fibonacci(" + i + ") = " + fibonacci(i));
      System.out.println("起動回数は" + count + "回");
    }
  }
  // フィボナッチ数列の 第 k 項を計算するメソッド
  public static int fibonacci(int k) {
    count++;
    if (k == 0) return 0;
    else if (k == 1) return 1;
    else
      return fibonacci(k - 1) + fibonacci(k - 2);
    }
  }
}
```

iが3のときの再帰呼び出しの連鎖とクラスフィールドcountの値の変化

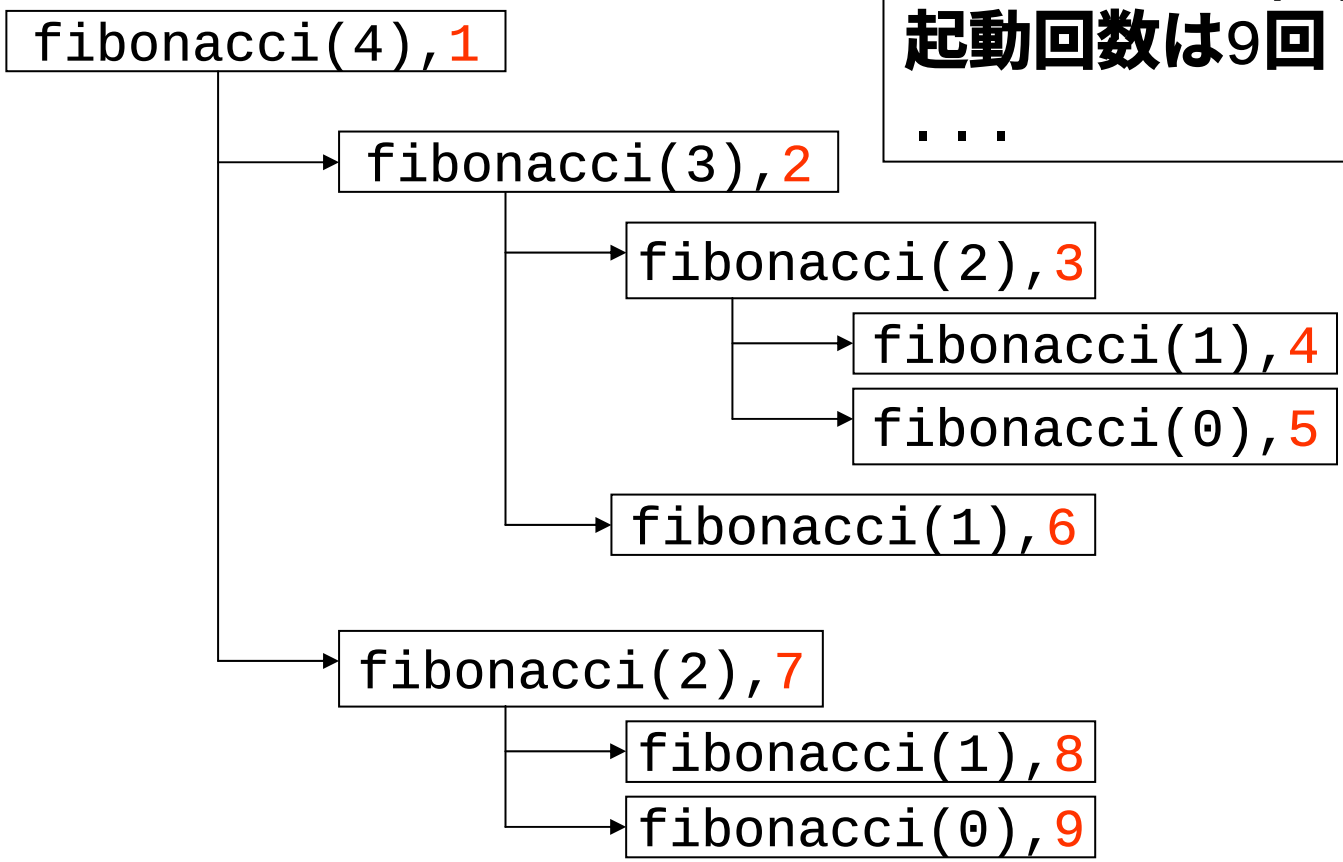
赤の数字はクラスフィールドcountの値




```
...
fibonacci(3) = 2
起動回数は5回
...
```

iが4のときの再帰呼び出しの連鎖とクラスフィールドcountの値の変化

```
...
fibonacci(4) = 3
起動回数は9回
...
```



iが0から10まで走ると

```
fibonacci(0) = 0
起動回数は1回
fibonacci(1) = 1
起動回数は1回
fibonacci(2) = 1
起動回数は3回
fibonacci(3) = 2
起動回数は5回
fibonacci(4) = 3
起動回数は9回

...

起動回数は67回
fibonacci(9) = 34
起動回数は109回
fibonacci(10) = 55
起動回数は177回
```

紙上演習

2006年夏の補講から