情報処理技法
(Javaプログラミング)2

第7回
メッセージ

人間科学科コミュニケーション専攻
白銀 純子

第7回の内容

- オブジェクト指向におけるメソッド
- メッセージ

前回の出席課題の回答

- アルゴリズムの良し悪しに関して、下記の文章の(ア)～(オ)を埋めなさい。

アルゴリズムの速さは、コンピュータで実行したときの秒・分単位の時間的な速さではなく、(ア)で表現される。(ア)は、アルゴリズムを実行する際の基本処理の回数である。アルゴリズムで扱うデータの個数を「N」として、N^2やN^3などの計算を必要とするアルゴリズムを(イ)アルゴリズム、N!や2^Nなどの計算を必要とするアルゴリズムを(ウ)アルゴリズムと呼ぶ。

速いアルゴリズムとわかりやすいアルゴリズムの関係は以下の通りである。

- 速いアルゴリズムは(エ)アルゴリズムである。
- (オ)アルゴリズムは遅いアルゴリズムである。

解答例:

(ア) 計算量
(イ) 多項式時間
(ウ) 指数時間
(エ) わかりにくい
(オ) わかりやすい

前回の復習

クラスとオブジェクト(1)

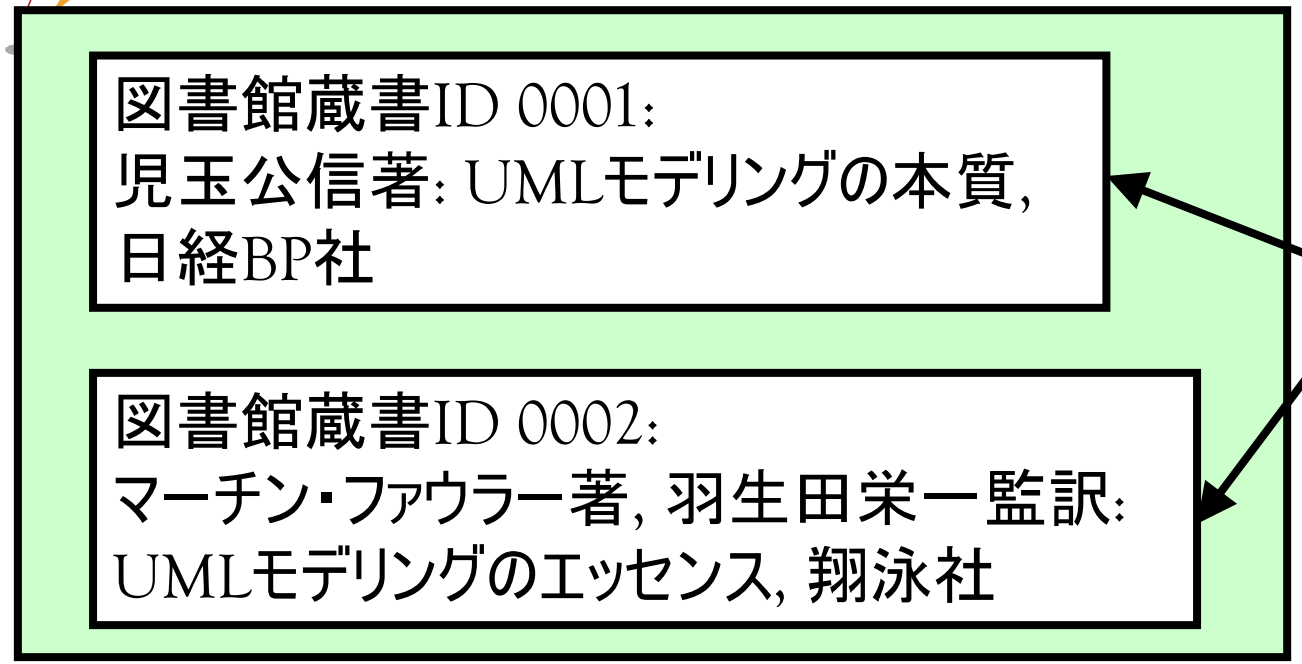
- クラス
  - 実物を分類したカテゴリ(実物の総称のような概念)
  - 名前を示されたとき、その概念にあてはまるものがいくつか存在するもの

    → 人や物を、持っている情報によって分類したもの
    Ex. 東京女子大学の学生

- オブジェクト
  - 1つ1つの具体的な実物
  - 名前を示されたとき、「これ」とそのものを特定できるもの

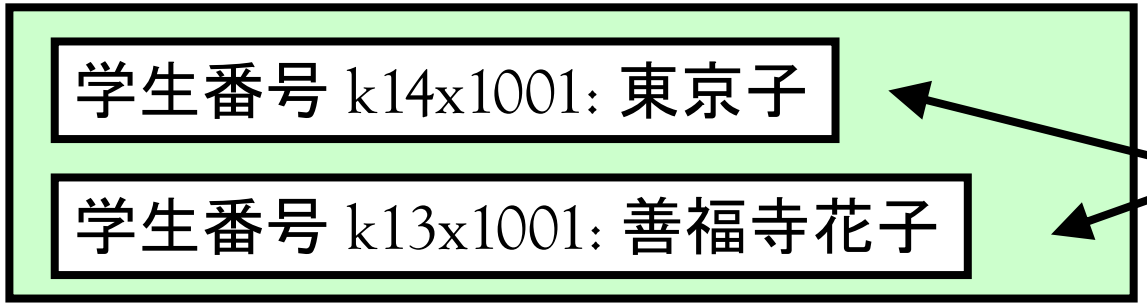
    → 「クラス」の分類に当てはまる、具体的な人や物
    Ex. 東京女子大学の学生の東京子さん

クラスとオブジェクト(2)



実物の本 =オブジェクト

「本」というカテゴリ(クラス)に分類

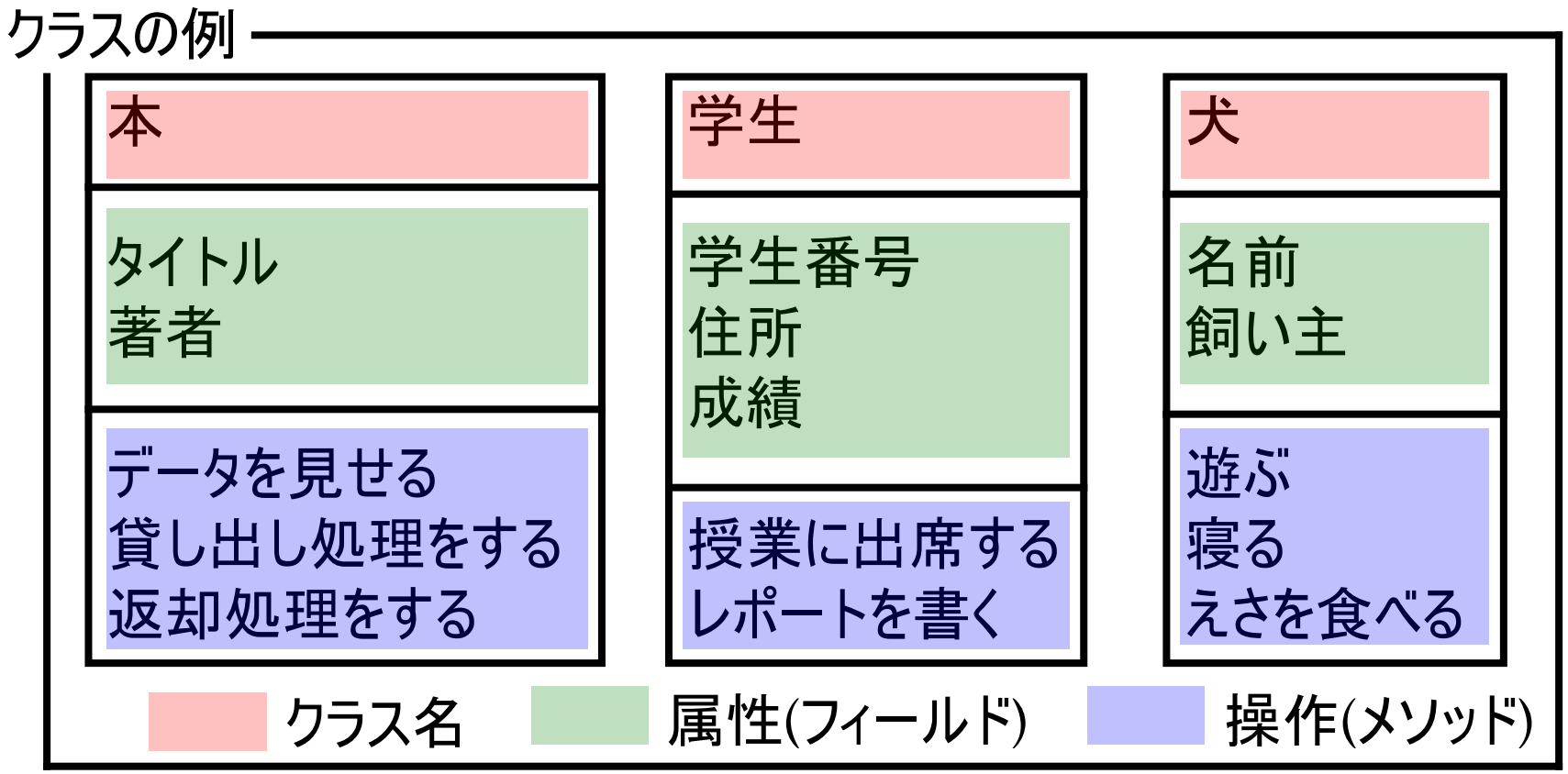


実物の学生 =オブジェクト

「学生」というカテゴリ(クラス)に分類

クラスとオブジェクト(3)

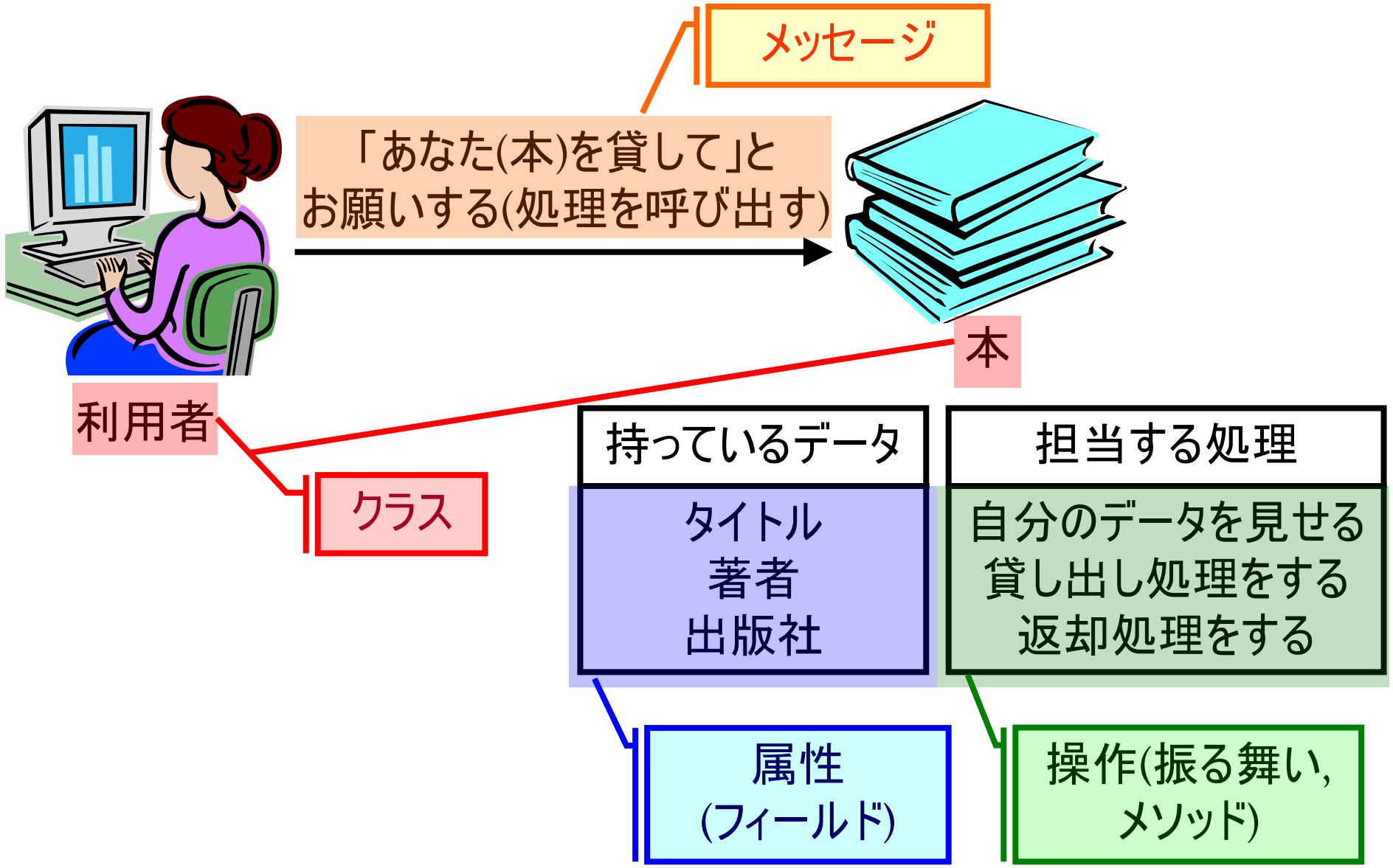
- クラス: 同じ属性と操作を持つオブジェクトの集合
  - 属性(フィールド):オブジェクトが持つ情報(データ)
  - 操作(振る舞い, メソッド): オブジェクトが担当する処理



クラスとオブジェクト(4)

- 1つのクラスにオブジェクトを所属させることができる
  - クラス: 実物を分類したカテゴリのようなもののため
- オブジェクト同士は、それぞれのクラスに定義された操作(処理)を呼び出す
  - 操作(処理)の呼び出しを「メッセージ」と呼ぶ
- メッセージを組み合わせてオブジェクト同士がコミュニケーションすることでプログラム全体が成り立つ

属性・操作・メッセージ(例)



プログラムでのクラスとオブジェクト

プログラムでしなければならないこと

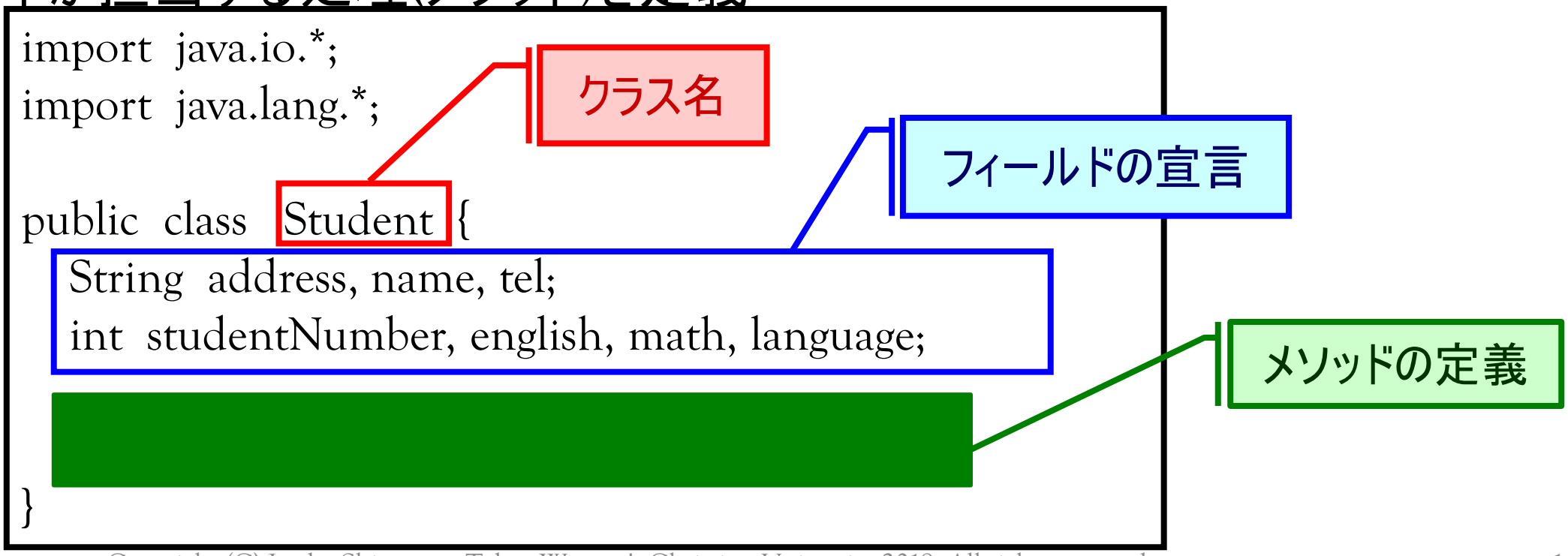
1. クラスを定義する
   - それぞれの「もの」について、内容を定義する
     - どのような名前か?
     - どのような情報(属性)を持っているか?
     - どのような操作(メソッド)を持っているか?
2. オブジェクトを作る
   - クラスに所属する個々のオブジェクトの情報の入れ物を作成
3. オブジェクトにデータを設定する
   - 2. で作ったオブジェクトに、具体的なデータを設定

原則

- データを定義するためのクラス(Javaファイル)を1つ作成
  - 処理のクラスとは別に作成
- 処理をするためのクラス(Javaファイル)を1つ作成
  - データ定義のクラスとは別に作成
- 処理のクラスの中で、データ定義のクラスのオブジェクトを作成
- 処理のクラスの中に、オブジェクトを使って、様々な処理を記述

1. クラスの定義のしかた

- これまでと同じ
  - 1ファイル1クラス
  - オブジェクトが持つデータ(フィールド)を変数として宣言
    - どのメソッドにも含まれない場所で宣言
  - オブジェクトが担当する処理(メソッド)を定義

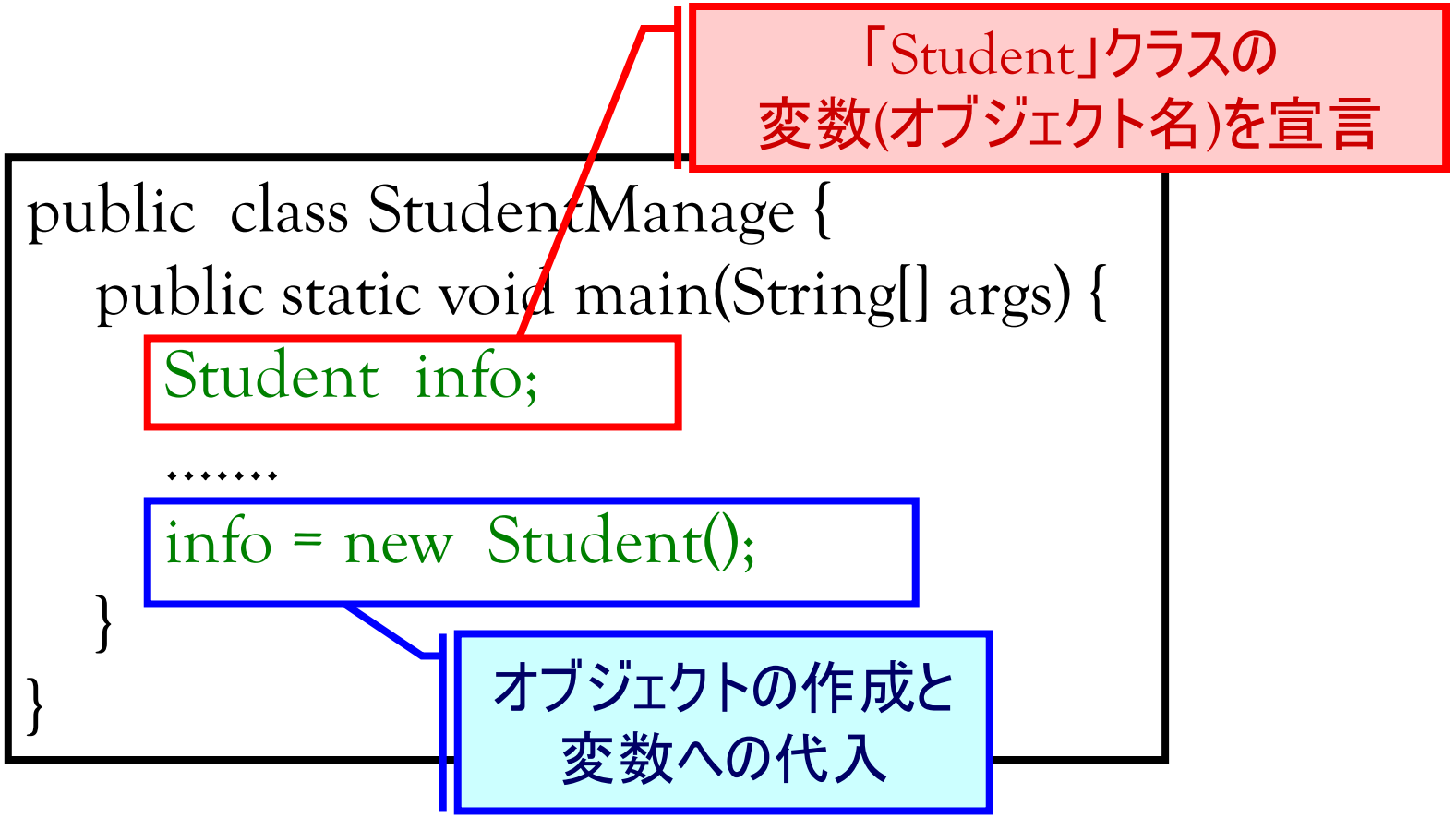


プログラムでしなければならないこと

1. クラスを定義する
   - それぞれの「もの」について、内容を定義する
     - どのような名前か?
     - どのような情報(属性)を持っているか?
     - どのような操作(メソッド)を持っているか?
2. オブジェクトを作る
   - クラスに所属する個々のオブジェクトの情報の入れ物を作成
3. オブジェクトにデータを設定する
   - 2. で作ったオブジェクトに、具体的なデータを設定

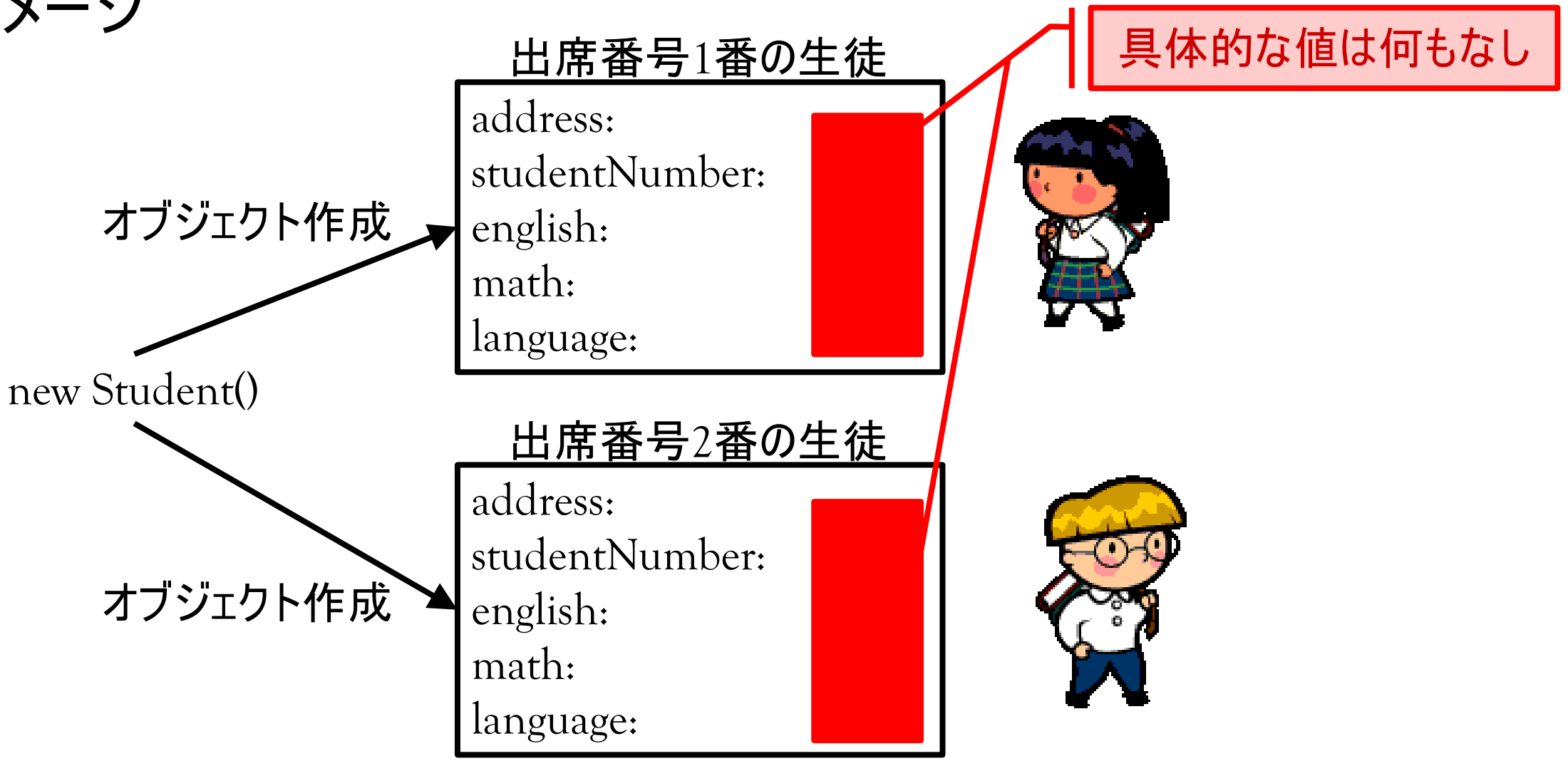
2. オブジェクトの作り方

- 「new クラス名()」でオブジェクトを作成し変数に代入
  - この作成・代入処理は、1. のクラスとは別のクラスのメソッド内で行う



「オブジェクトを作る」とは?

- 具体的な情報が何も設定されていない、情報の入れ物を作る、というイメージ



「オブジェクト」が複数ある場合

- 高校の生徒: 何人も存在

StudentManage.java

```
public  class StudentManage {
    public static void main(String[] args) {
        Student  info;
        ……
        info = new  Student();
    }
}
```

これだと、1人分の情報だけ

オブジェクトを配列またはArrayListにする

複数のオブジェクトの扱い～配列～(1)

- オブジェクト: プログラムでの表記は変数と同じ
  = これまでのintやStringと同様に配列の宣言が可能

```
public  class StudentManage {
    public static void main(String[] args) {
        Student[]  info = new  Student[50];
        ……..
        info[0] = new  Student();
        info[1] = new  Student();
        ……..
    }
}
```

「Student」クラスの
オブジェクトを50個分宣言

複数のオブジェクトの扱い～配列～(2)

- これまでと同様、「オブジェクト名[添え字] = new クラス名();」で作成
  - 配列で扱う個々のオブジェクトの作成を忘れないこと

```
public  class StudentManage {
    public static void main(String[] args) {
        Student[]  info = new  Student[50];
        ……..
        info[0] = new  Student();
        info[1] = new  Student();
        ……..
    }
}
```

オブジェクトを1つ1つ作成(for文やwhile文でまとめて作成してもOK)

[ ]と( )の違いに注意!

- [ ]: 配列を表す
  - Student info[ ] = new Student[30];
    ←変数「info」を、Studentクラスの30個の要素を持つ**配列として宣言**
  - Student info = new Student( );
    ←変数「info」に、Studentクラスの変数として宣言し、**オブジェクトを代入**

「new Student...」と書いていても、意味が全く違うので注意!
- オブジェクトを配列にするときは、配列としての宣言と、
  各要素へのオブジェクトの代入が必要

```
Student[ ] info = new Student[30]; // infoを30個の要素を持つ配列として宣言
info[0] = new Student( ); // info[0]にオブジェクトを代入
info[1] = new Student( ); // info[1]にオブジェクトを代入
…
```

プログラムでしなければならないこと

1. クラスを定義する
   - それぞれの「もの」について、内容を定義する
     - どのような名前か?
     - どのような情報(属性)を持っているか?
     - どのような操作(メソッド)を持っているか?
2. オブジェクトを作る
   - クラスに所属する個々のオブジェクトの情報の入れ物を作成
3. オブジェクトにデータを設定する
   - 2. で作ったオブジェクトに、具体的なデータを設定

オブジェクトの利用(値の代入と参照)(1)

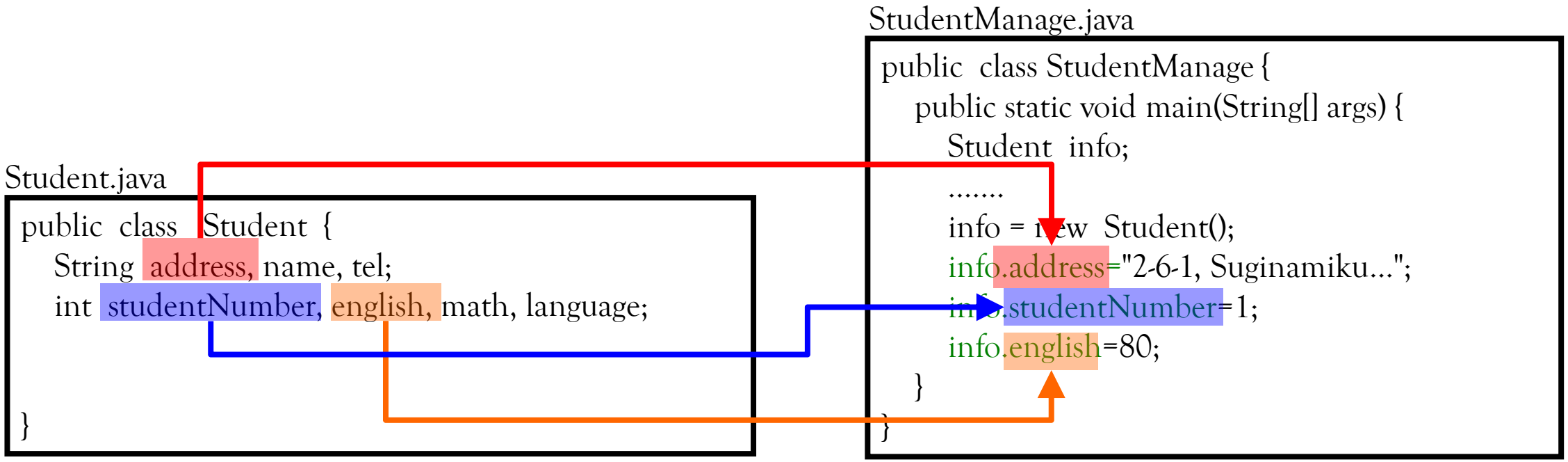
- オブジェクトの作成後、フィールドに値を代入可能
  - 「オブジェクト名.フィールド名」で普通の変数と同様に扱う
    - 「new」として、オブジェクトを作成したクラスのメソッド内で、「オブジェクト名.フィールド名」という変数を利用できる

```
public  class StudentManage {
    public static void main(String[] args) {
        Student  info;
        .......
        info = new  Student();
        info.address="2-6-1, Suginamiku...";
        info.studentNumber=1;
        info.english=80;
    }
}
```

フィールドに
値を代入

オブジェクトの利用(値の代入と参照)(2)

- 「オブジェクト名.フィールド名」で、「フィールド名」として使えるのは
  1. で定義したクラスのフィールドの変数
  - 「オブジェクト名.フィールド名」で、「オブジェクト」「の(.)」「フィールド名」という意味



オブジェクトの配列化～代入～(1)

- 「オブジェクト名[添え字].フィールド名」で、通常の変数と同様に扱う

```
public  class StudentManage {
    public static void main(String[] args) {
        Student[]  info = new  Student[50];
        ……..
        info[0] = new  Student();
        ……..
        info[0].address="2-6-1, Suginamiku...";
        info[0].studentNumber=1;
        info[0].english=80;
        ……..
    }
}
```

オブジェクトのフィールドに1つ1つ値を代入

オブジェクトの配列化～代入～(2)

- 「配列の要素.フィールド名」で、個々のオブジェクトの情報を表現

出席番号1番の生徒(info[0])

住所: info[0].address
出席番号: info[0].studentNumber
英語の得点: info[0].english
数学の得点: info[0].math
国語の得点: info[0].language

出席番号2番の生徒(info[1])

住所: info[1].address
出席番号: info[1].studentNumber
英語の得点: info[1].english
数学の得点: info[1].math
国語の得点: info[1].language

処理クラスの中で
変数として利用

オブジェクトの配列化～代入～(2)

- フィールドに値を入れることにより、各オブジェクトの固有のデータが設定

```
public  class StudentManage {
    public static void main(String[] args) {
        .......
        info[0].address="2-6-1, Suginamiku...";
        info[0].studentNumber=1;
        info[0].english=80;
        .......
        info[1].address="1-1-1, Kichijoji...";
        info[1].studentNumber=2;
        info[1].english=93;
    }
}
```

出席番号1番の生徒

address: 2-6-1, Suginamiku...
studentNumber: 1
english: 80
math:
language:

出席番号2番の生徒

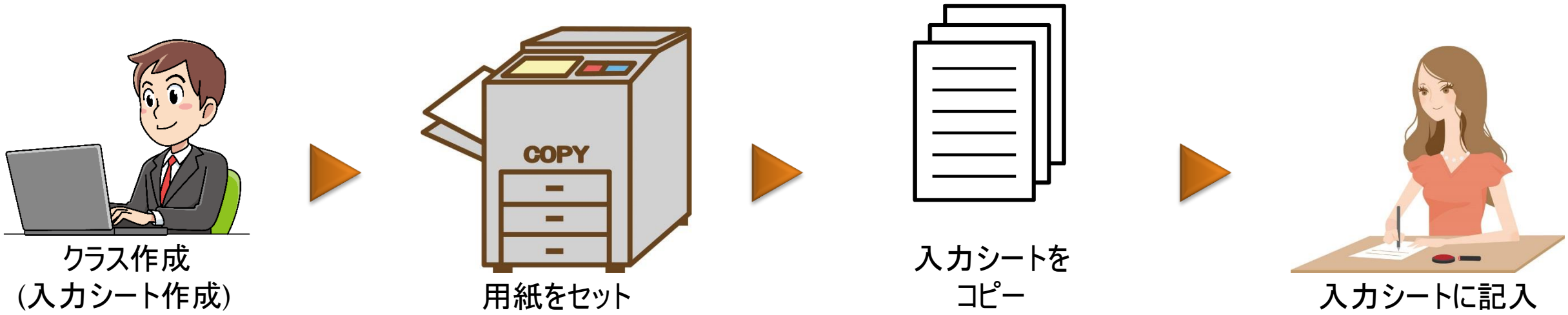
address: 1-1-1, Kichijoji...
studentNumber: 2
english: 93
math:
language:

オブジェクトの配列化～利用～

- オブジェクトを配列にしたときも、添え字の考え方はこれまでと全く同じ
  - 添え字は0から数え始める
  - 0～[宣言した数-1]の番号の添え字を利用できる
    - ..., -3, -2, -1や、[宣言した数], [宣言した数+1], [宣言した数+2], ...は使えない
  - 高校の生徒などの場合、添え字と出席番号を対応させると扱いやすい
    - Ex. 出席番号1番の生徒は添え字0, 出席番号2番の生徒は添え字1, ...

クラスと配列のオブジェクトのイメージ

- クラス作成～フィールドへの値の代入は、個人情報の入力シートの作成をして、シートに入力するまでの流れのイメージ
  - クラス作成: 個人情報の入力シートの作成
  - オブジェクトの変数(配列)宣言: コピー機に用紙をセット
  - オブジェクトの作成(配列): コピー機で個人情報の入力シートをコピー
  - フィールドに値を代入: 1人1人がシートに記入



メッセージのやり取り

メッセージ

- メッセージ = あるクラスで定義されたメソッドを呼び出すこと
- 「オブジェクト名.メソッド」(または「クラス名.メソッド」)の形式で呼び出し
  - ただし、メソッドを定義している同じクラス内で呼び出すときは、オブジェクト名やクラス名は省略
  - Ex1. str.substring(m, n)
    - Stringクラスに定義されている「substring」というメソッドを呼び出し
      (strはStringクラスのオブジェクト = String型の変数)
  - Ex2. Integer.parseInt(str)
    - Integerクラスに定義されている「parseInt」というメソッドを呼び出し
      (strはStringクラスのオブジェクト = String型の変数)

メッセージのやり取りをするには

1. 各クラスでメソッドを定義する
2. オブジェクト同士でメソッドを呼び出しあう

メソッドの定義

- クラスに関連づけるメソッドとオブジェクトに関連づけるメソッドの2種類
(内容の定義はこれまでと全く同じ)

クラスに関連づけるメソッドの定義のテンプレート

```
public　static　戻り値のデータ型　メソッド名(引数) {
    メソッドでの処理内容
    return　処理結果;
}
```

オブジェクトに関連づけるメソッドの定義のテンプレート

```
public　戻り値のデータ型　メソッド名(引数) {
    メソッドでの処理内容
    return　処理結果;
}
```

違い:「static」キーワードがついているかいないか

「static」のあるなし(1)

- 「static」がついているメソッド(クラスメソッド)
  - これまで作ってきたメソッド
  - 「クラス名.メソッド」の形式で呼び出し可能(「オブジェクト名.メソッド」の形式でも可能)
  - staticつきのメソッド内部で、同じクラスで定義されているメソッドを呼び出すときは、そのメソッドにもstaticが必要
    - mainメソッドから呼び出すときは、「static」がついている必要
  - メソッドを定義しているクラスのインスタンス変数を利用することは不可能
    (クラス変数は利用可能)

オブジェクトによって処理結果の変わらないメソッドはstaticをつけて良い
(つけないメソッドにしても良い)

「static」のあるなし(2)

- 「static」がついていないメソッド(インスタンスメソッド)
  - 必ず「オブジェクト名.メソッド」の形式で呼び出し(「クラス名.メソッド」の形式では呼び出し不可能)
  - staticなしのメソッド内部で、同じクラスで定義されているメソッドを呼び出すときは、そのメソッドはstaticはついていても、ついていなくても良い
  - staticなしのメソッド内部で、同じクラスで定義されているフィールドは、インスタンス変数・クラス変数とも利用可能

オブジェクトによって処理結果の変わるメソッド(同じクラスで定義されているstaticなしのフィールドを処理に使うなど)は、必ずstaticなし

mainメソッド

- 「この部分を最初に実行する」という意味のメソッド
  - クラスメソッドの一種
    - 「public static void main(String[] args) {」のメソッド
  - Javaでは、プログラムを実行したときに、まず最初にmainメソッドの「{」と「}」の間に書かれている処理を実行
    - 複数のクラス作成するときは、mainメソッドを作成するのは1つのクラスのみ
    - 複数のクラスを使ってプログラムを実行するときは、「java」コマンドで指定するクラスは、メインメソッドを持っているクラス

オブジェクト同士でのメソッドの呼び出し

- あるクラス(名前: ClassA)で定義されているメソッドを...
  - 別のクラスから呼び出す場合

    メソッドがインスタンスメソッドの場合: ClassAのオブジェクト名.メソッド

    メソッドがクラスメソッドの場合: ClassA.メソッド

  - 同じクラス(ClassA)から呼び出す場合:「オブジェクト名.」や「クラス名.」は不要
    - メソッド名 + 引数のみでOK

メソッドの定義とメッセージでの処理の流れ(例)(1)

ProductManages.java

```
// 割引率と税率を設定して、商品の金額計算
public class ProductManage {
  public static void main(String[] args) {
    int amount;

    Product pro = new Product();
    // 割引率: 0.2 (2割引)
    amount = pro.calcAmount(0.2, 0.08);
  }
}
```

Product.java

メソッドの定義

```
// 商品の値段管理
public class Product {
  int  price;

  // 商品の金額計算(discount: 割引率, tax: 税率)
  public int calcAmount(double discount, double tax) {
    double doubleAmount;
    int intAmount;

    doubleAmount = (double) price * (1 - discount) * (1 + tax);
    intAmount = (int) doubleAmount;
    return intAmount;
  }
}
```

メソッドの定義とメッセージでの処理の流れ(例)(2)

引数には具体的な値または変数を書く
(引数の順番は、メソッドを作ったときの順番と同じに)

ProductManages.java

```
// 割引率と税率を設定して、商品の金額計算
public class ProductManage {
  public static void main(String[] args) {
    int amount;

    Product pro = new Product();
    // 割引率: 0.2 (2割引)
    amount = pro.calcAmount(0.2, 0.08);
  }
}
```

Product.java

```
// 商品の値段管理
public class Product {
  int  price;

  // 商品の金額計算(discount: 割引率, tax: 税率)
  public int calcAmount(double discount, double tax) {
    double doubleAmount;
    int intAmount;

    doubleAmount = (double) price * (1 - discount) * (1 + tax);
    intAmount = (int) doubleAmount;
    return intAmount;
  }
```

calcAmountというメソッドを呼び出す

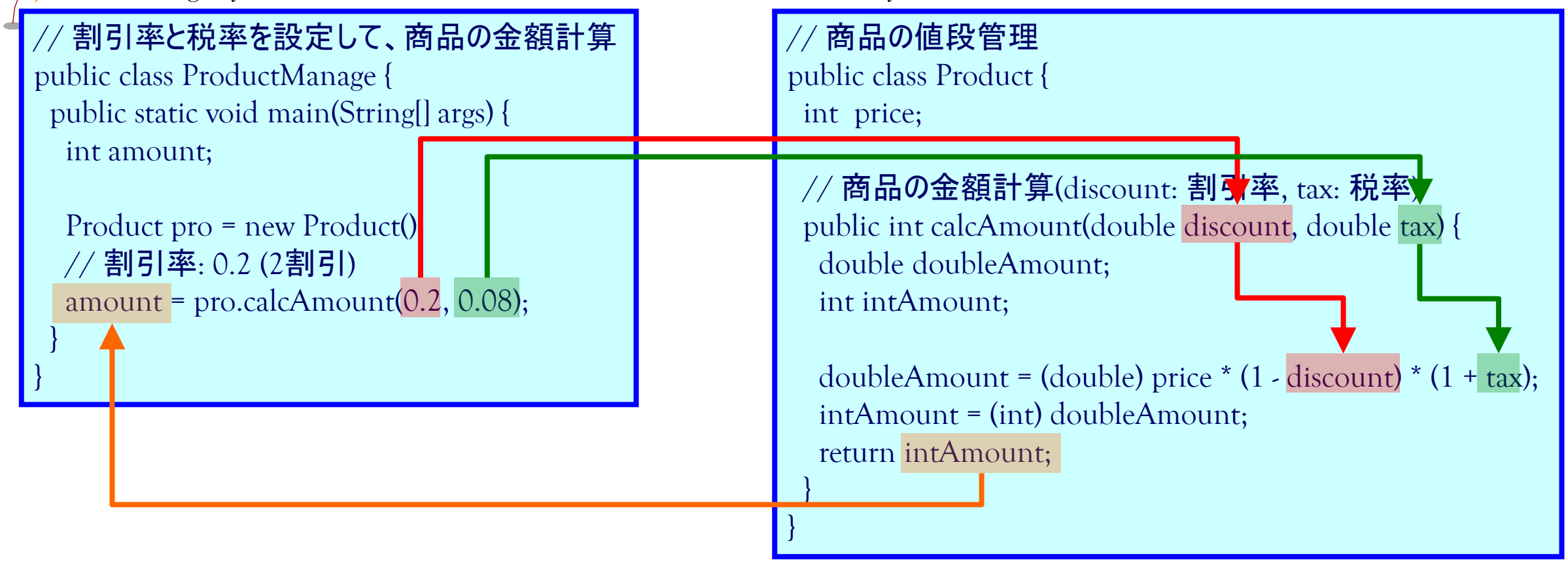
メソッドの定義とメッセージでの処理の流れ(例)(3)

ProductManages.java

```
// 割引率と税率を設定して、商品の金額計算
public class ProductManage {
  public static void main(String[] args) {
    int amount;

    Product pro = new Product();
    // 割引率: 0.2 (2割引)
    amount = pro.calcAmount(0.2, 0.08);
  }
}
```

Product.java

```
// 商品の値段管理
public class Product {
  int  price;

  // 商品の金額計算(discount: 割引率, tax: 税率)
  public int calcAmount(double discount, double tax) {
    double doubleAmount;
    int intAmount;

    doubleAmount = (double) price * (1 - discount) * (1 + tax);
    intAmount = (int) doubleAmount;
    return intAmount;
  }
}
```



処理の流れ

1. mainメソッドで、calcAmountメソッドの引数に0.2と0.08を指定する
2. 指定された0.2と0.08というデータがcalcAmountメソッドの引数の変数「discount」と「tax」にそれぞれ代入される
3. calcAmountメソッド内で引数の値を使って計算され、変数intAmountに結果が代入される
4. calcAmountメソッドの変数intAmountの値がmainメソッドの変数amountに代入される

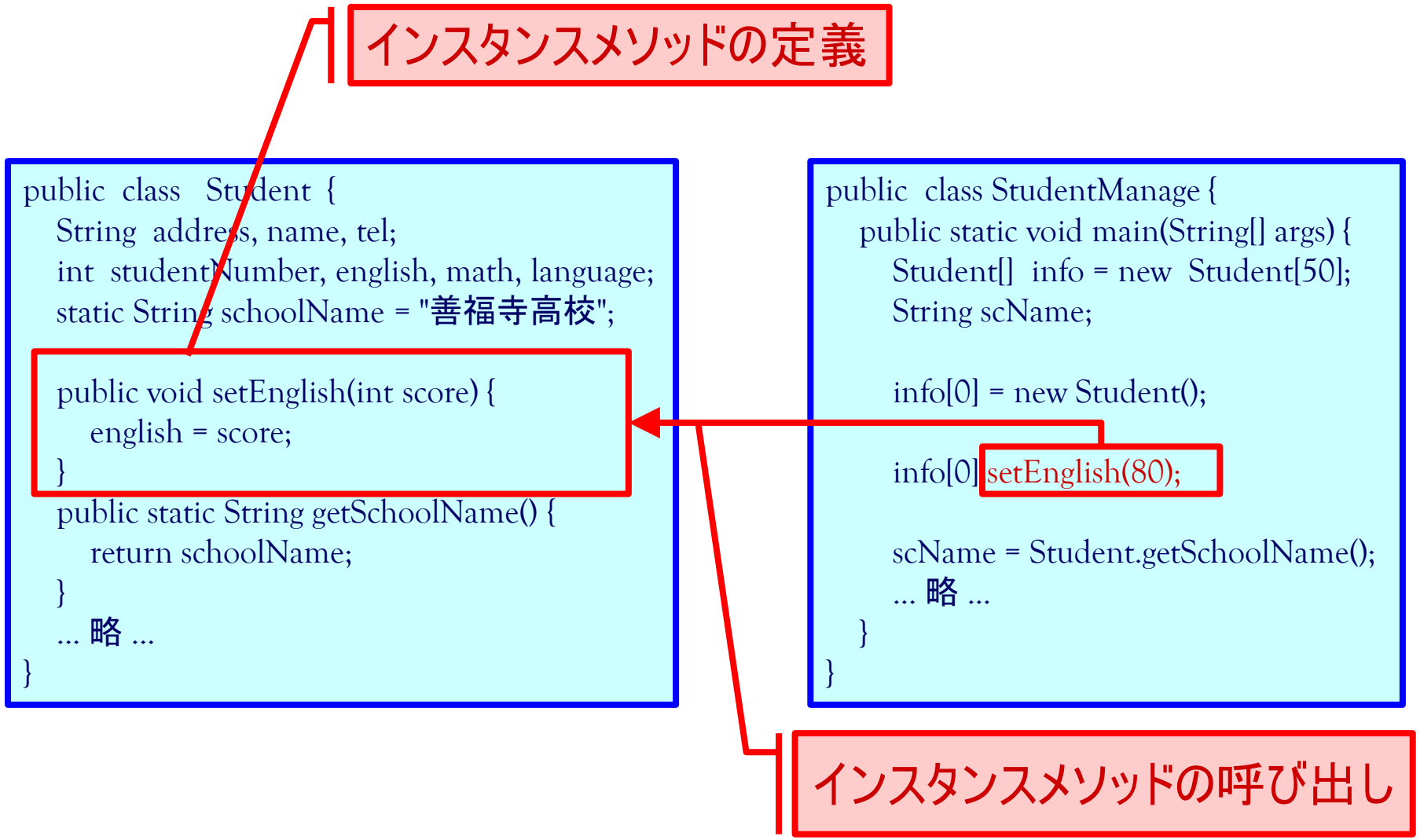
メソッドの呼び出し(例)(1)

```
public  class   Student  {
    String  address, name, tel;
    int  studentNumber, english, math, language;
    static String schoolName = "善福寺高校";

    public void setEnglish(int score) {
        english = score;
    }
    public static String getSchoolName() {
        return schoolName;
    }
    … 略 …
}
```

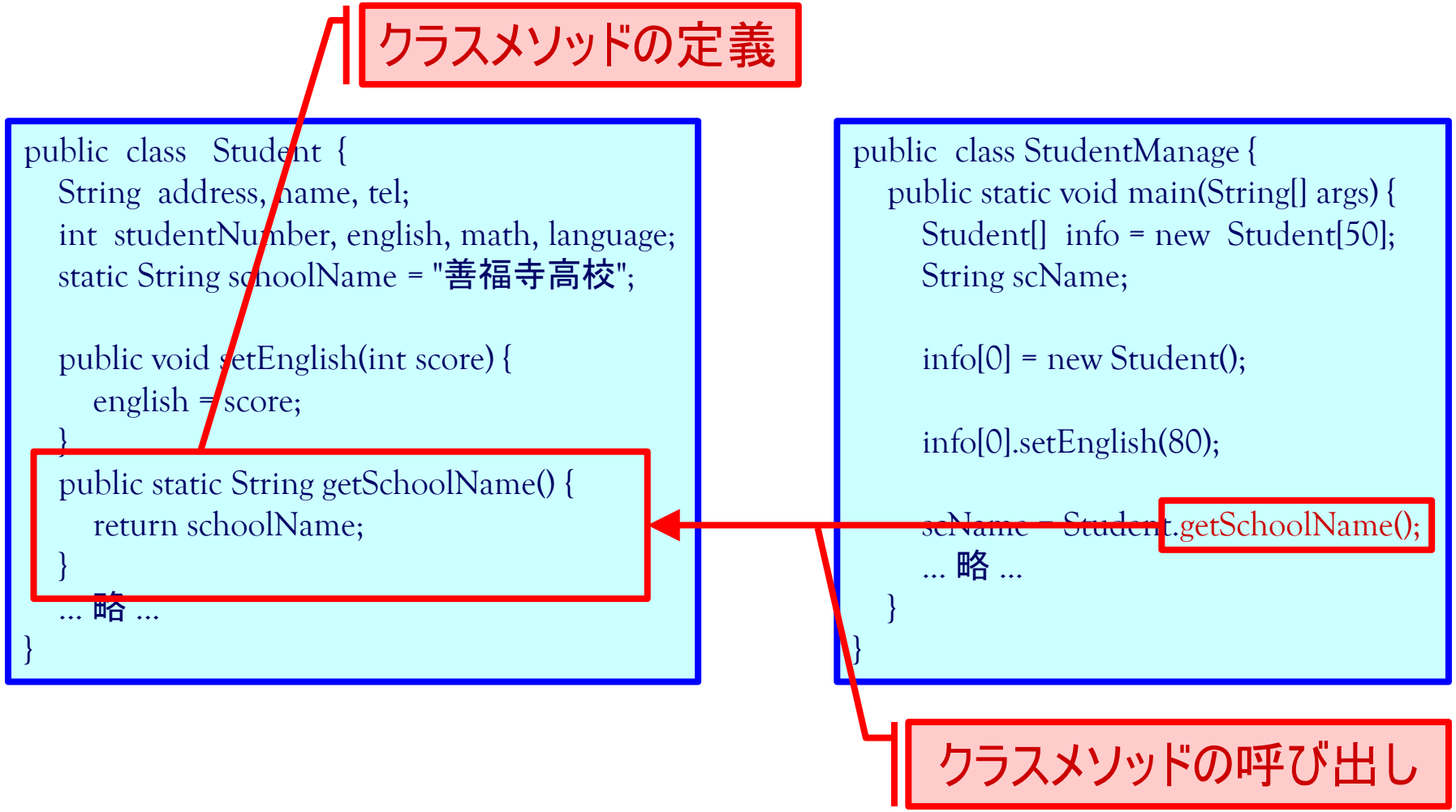
```
public  class StudentManage {
    public static void main(String[] args) {
        Student[]  info = new  Student[50];
        String scName;

        info[0] = new Student();

        info[0].setEnglish(80);

        scName = Student.getSchoolName();
        … 略 …
    }
}
```

※クラス変数は、宣言と同時に値を代入してOK

メソッドの呼び出し(例)(2)



メソッドの呼び出し(例)(3)



コンストラクタ

特殊なメソッド～コンストラクタ～(1)

- コンストラクタ: オブジェクトを作成すると同時に実行される特殊なメソッド
- 通常のメソッドとの違い
  - 名前が必ずクラス名と同じ
  - 戻り値のデータ型の定義なし
  - 戻り値を返すためのreturn文はなし
  - 「static」キーワードをつけることは不可
- 通常のメソッドと同じもの
  - 引数を定義(なくても良い)
  - 処理内容の記述方法は同じ

特殊なメソッド～コンストラクタ～(2)

- 定義方法:「public クラス名(引数のリスト){...}」
  - 引数のリストはなくてもよい
  - 「{」から「}」の間に、オブジェクトを作成すると同時に実行される処理内容を記述する
  - オブジェクトを作成するとき、「new クラス名() ;」とするのは、コンストラクタを呼び出している、ということ
- コンストラクタでよく処理するもの
  - フィールドの値の設定
  - GUIを扱うクラスの場合、GUIのウィンドウの作成処理

コンストラクタの例

クラスの定義

```
public  class  Student {
   String  number, circle;
   int score;
   public  Student(String  n,  int  s,  String  c) {
      number = n;
      score = s;
      circle = c;
   }
}
```

コンストラクタ

オブジェクト作成例

```
Student  member;
member = new Student("k17x1001", 80, "オーケストラ");
```

```
Student  member;
member = new Student();
member.number = “k17x1001”;
member.score = 80;
member.circle = "オーケストラ"
```

同じ意味

コンパイルと実行

コンパイルと実行のしかた

- コンパイル
  - 「javac」の後に、ファイル名をスペースでつなげて複数のファイルをコンパイル

```
% javac  StudentManage.java  Student.java
```

  - または、「*」でそのフォルダに保存されているJavaファイルすべてをコンパイル
    - プログラムに関係ないJavaファイルもコンパイルされる。関係ないJavaファイルにコンパイルエラーがあれば、コンパイルが完了しないので注意

```
% javac  *.java
```

- 実行
  - 「java」の後に、「public static void main」が書かれているファイル名(拡張子なし)を書く

```
% java  StudentManage
```

やってみよう![1]

- 下記の2つのクラスを持つプログラム
  - 1つ目のクラス: 生徒クラス
    - 名前と出席番号、5教科の試験の得点を入れるフィールドを持つ
    - 5教科の平均点を計算し、戻り値とするメソッドを持つ
  - 2つ目のクラス: 処理クラス
    - 生徒クラスのオブジェクトに5教科の試験の得点を設定する
    - 生徒クラスのメソッドを使い、5教科の試験の平均点を計算する
- ケーキクラスを作成し、ケーキの名前と値段をコンストラクタの引数として与え、コンストラクタの中でケーキの名前と値段をフィールドに代入するプログラム
  - 代入した結果を標準出力に出力すること

やってみよう![2]

- 下記の2つのクラスを持つプログラムを作ること
  - 1つ目のクラス: 友達の情報を管理し、挨拶をするクラス
  - フィールド: 友達の名前
    - メソッド: 「こんにちはxxさん! 私はyyです。よろしくお願いします。」と標準出力で出力(xxは引数として与え、yyはフィールドの値)
      - 引数: 自分の名前(xx)
      - 戻り値: なし
  - 2つ目のクラス: 1つ目のクラスのオブジェクトを作成し、メソッドを呼び出すクラス
    - 自分の名前を標準入力で入力
    - 1つ目のクラスのオブジェクトを作成し、友達の名前を設定(標準入力などはしなくて良い)
    - 1つ目のクラスのメソッドを呼び出し
    - 呼び出したメソッドの引数は、標準入力で入力した自分の名前

やってみよう![3]

- 下記の2つのクラスを持つプログラムを作ること
  - 1つ目のクラス: おこづかい帳の収入と支出を管理するクラス
    - フィールド: 収入の金額, 支出の金額(支出の金額は配列)
    - メソッド: 収入に対する支出の割合を、パーセンテージで計算
      - 引数: なし
      - 戻り値: 計算したパーセンテージの数
  - 2つ目のクラス: 1つ目のクラスのオブジェクトを作成し、メソッドを呼び出すクラス
    - 収入の金額と支出の金額を標準入力で入力
      - 収入は1つだけ入力
      - 支出は、「End」と入力されるまで、何回でも入力
    - 1つ目のクラスのオブジェクトを作成し、入力された収入と支出の金額を設定
    - 1つ目のクラスのメソッドを呼び出して、収入に対する支出の割合を計算し、標準出力で出力