情報処理技法
(Javaプログラミング)2

第4回
オブジェクト指向って?

人間科学科コミュニケーション専攻
白銀 純子

第4回の内容

- プログラミングの種類
- オブジェクト指向とは?

前回の出席課題の解答

- このプログラムは、自分の名前と相手の名前を入力し、挨拶を出力するプログラムです。(処理)の部分にどのような処理が入るか、下記の選択肢から最も適切なものを選びなさい。

プログラム

```
public static void sayHello(String you, String me) {
  (処理)
}

public static void main(String[] args) {
  String you, me;
  try {
    BufferedReader br = new BufferedReader(new InputStreamReader(System.in));

    // 自分の名前の入力
    System.out.print("あなたの名前を入力してください > ");
    me = br.readLine();

    // 挨拶の相手のの名前の入力
    System.out.print("挨拶をする人の名前を入力してください > ");
    you = br.readLine();

    // 挨拶処理
    sayHello(you, me);
  }
  catch(IOException e) {
    System.out.println("標準入力ができませんでした");
  }
}
```

選択肢

```
(a)
  String message;
  message = "Hello, " + you + "! My name is " + me + "!";
  return message;

(b)
  System.out.println("Hello, " + you + "!");
  System.out.println("My name is " + me + "!");

(c)
  String message;
  message = "Hello, " + you + "! My name is " + me + "!";

(d)
  String message;
  message = "Hello, " + you + "! My name is " + me + "!";
  return message;
  System.out.println(message);
```

解答: b

プログラミングの種類

- 関数型言語
- 手続き型
- オブジェクト指向言語

関数型言語

- 数学的な関数のみをもとにして記述するプログラム言語
  - 一度変数に値が与えられれば、その変数の値は変化しない
  - 計算結果を引数とする、関数呼び出しのみで計算を行う
  - プログラミング言語: Lisp, Schemeなど

手続き型言語

- 記述された命令を上から順に実行していくプログラム言語
  - 処理の結果に応じて変数の値が変化
  - プログラミング言語: C言語, BASIC, Pascalなど

オブジェクト指向言語

- 「もの」と「もの」との関係に重点を置いて記述するプログラミング言語
  - ある「もの」に対して、それが持つ情報と、その「もの」が行う作業を記述する
  - ある「もの」と別の「もの」とのコミュニケーションを記述することで、プログラムを動作させる
  - プログラミング言語: SmallTalk, C++, C#など

Javaは?

- オブジェクト指向言語
- これまでの言語にはない、完全なオブジェクト指向を実現した言語
- 「Write Once, Run Anywhere」(一度記述すればどこでも動作する)がキャッチコピー
  - 一度記述すれば、OS等の環境が異なるコンピュータでもプログラムは動作する
  - 他のプログラミング言語では、OS等が違うとコンパイル・実行ができないこともある

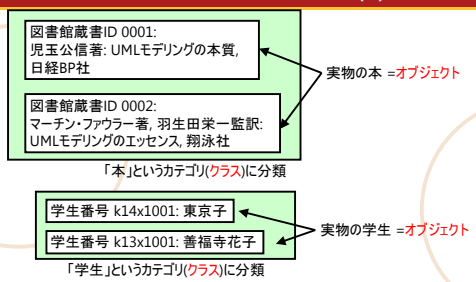
オブジェクト指向の基礎

クラスとオブジェクト(1)

- オブジェクト指向: 「もの」を中心してソフトウェアを構築する考え方
  - 「もの」: オブジェクト(インスタンスとも)
    - 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
  - 「もの」の分類: 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

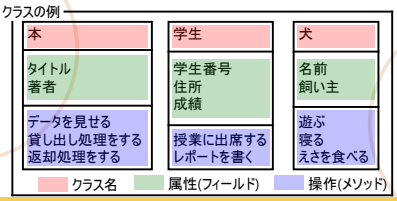
※「オブジェクト」と「インスタンス」は厳密にはちがうもの

クラスとオブジェクト(2)



クラスとオブジェクト(3)

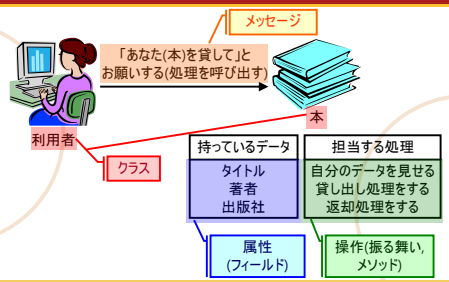
- クラス: 同じ属性と操作を持つオブジェクトの集合
  - 属性(フィールド):オブジェクトが持つ情報(データ)
  - 操作(振る舞い, メソッド): オブジェクトが担当する処理



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

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

```
import java.io.*;
import java.lang.*;

public class Student {
    String address, name, tel;
    int studentNumber, english, math, language;

}
```

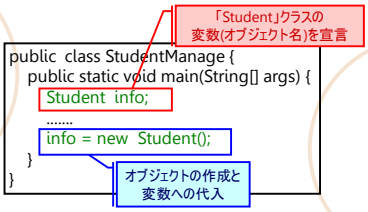
クラス名
フィールドの宣言
メソッドの定義

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

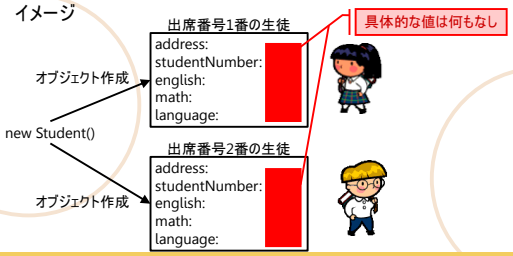
2. オブジェクトの作り方

- 「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



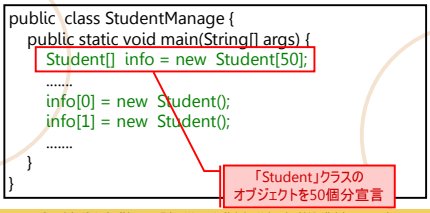
「オブジェクトを作る」とは?

- 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



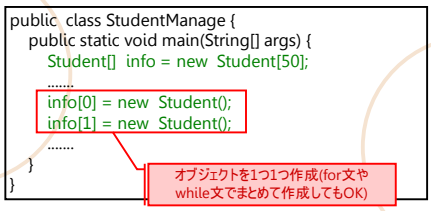
複数のオブジェクトの扱い～配列～(1)

- 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



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



「static」キーワード

- フィールドの変数に「static」というキーワードをつけて宣言することがある
  - Ex1. static String schoolName;
  - Ex2. static int classNumber;
- staticなしのフィールド(インスタンス変数)
  - オブジェクトごとに値が異なるフィールドを表現するために利用
    - Ex. 1人1人の生徒の住所や電話番号、試験の成績など
- staticつきのフィールド(クラス変数)
  - どのオブジェクトも値が同じであるフィールドを表現するために利用
    - Ex. 学校の名前など

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)

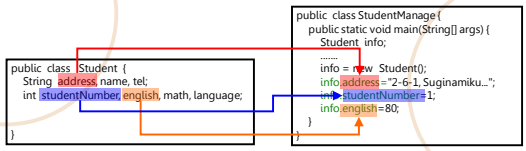
- オブジェクトの作成後、フィールドに値を代入可能
  - 「オブジェクト名.フィールド名」で普通の変数と同様に扱う
    - 「new」として、オブジェクトを作成したクラスのメソッド内で、「オブジェクト名.フィールド名」という変数を利用できる

```
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)

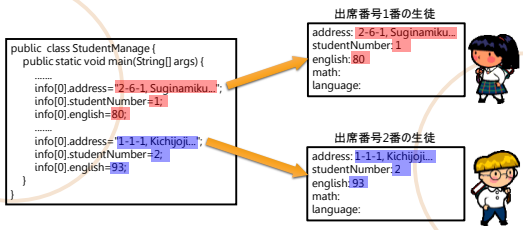
- 「オブジェクト名[添え字].フィールド名」で、通常の変数と同様に扱う

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

- 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



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, ...

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  返り値のデータ型  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  返り値のデータ型  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

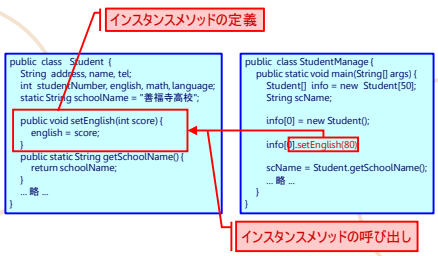
メソッドの呼び出し(例)(1)

```
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 ...
}
```

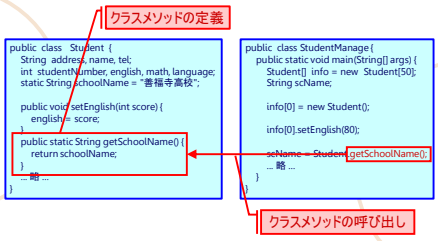
```
public class StudentManage {
   public static void main(String[] args) {
      Student[] info = new Student[50];
      String scName;

      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
  - 返り値のデータ型の定義なし
  - 返り値を返すための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("k13x1001", 80, "オーケストラ");
```

```
Student member;
member = new Student();
member.number = "k13x1001";
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!

- 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、返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