メディアプロジェクト演習１　参考資料

- Javaとは
- JavaScript と　Java言語の違い
- オブジェクト指向
- コンストラクタ
- サーブレット

※参考文献

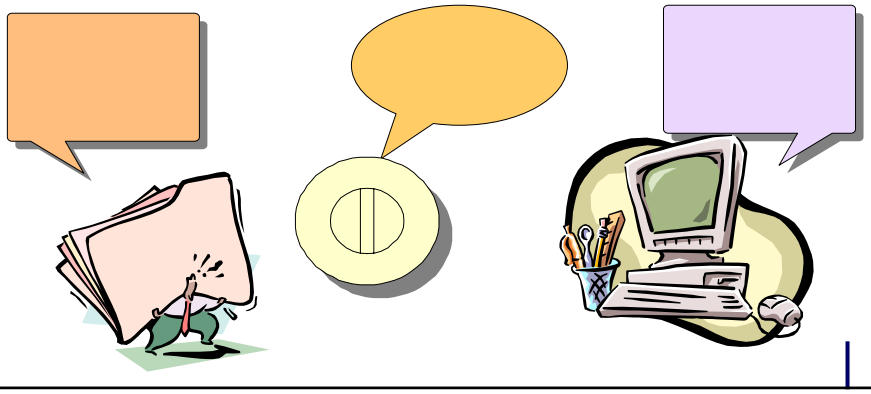
高橋麻奈:「やさしいJava（第3版）」, ソフトバンククリエイティブ（2,625円）

はじめに

- プログラミング言語とは？
- オブジェクト指向とは？
- Java言語とは？
- JavaとJavaScriptの違いとは？

人間の言葉に近い言葉を翻訳してコンピュータに伝える

- **プログラミング言語**　人間の言葉に近い言葉でコンピュータに作業内容を伝える言語
- **コンパイラ、インタプリタ**　プログラミング言語で書かれた作業内容を機械語に翻訳するソフトウェア



Javaとは

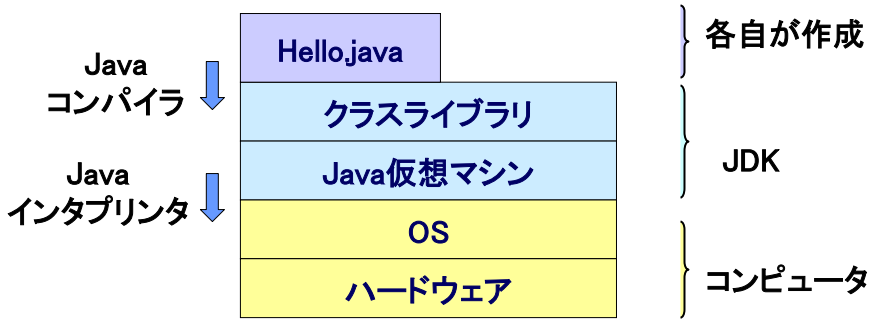
- プログラミング言語の一種
- Javaコンパイラ と Javaインタプリタを利用して機械語に翻訳する
- 特徴
  - 機種依存の少ない言語(マルチプラットフォーム)
    - 動作環境(OS)を選ばない
    - Java仮想マシンを内蔵している様々な機器で動作
  - オブジェクト指向
  - ネットワークでの利用を想定した仕様

Javaの範囲

- 広い範囲において開発に利用
  - 携帯電話のゲーム機能もJavaで作成
  - WebサービスもJavaで提供されている
  - 家電用の組込機器向けプログラミング言語
    - 近年、インターネットに接続できる電子レンジまで発売

Javaの実行環境

- “Hello”の出力を行うJavaのソフト実行環境
  （ファイル名:Hello.java）



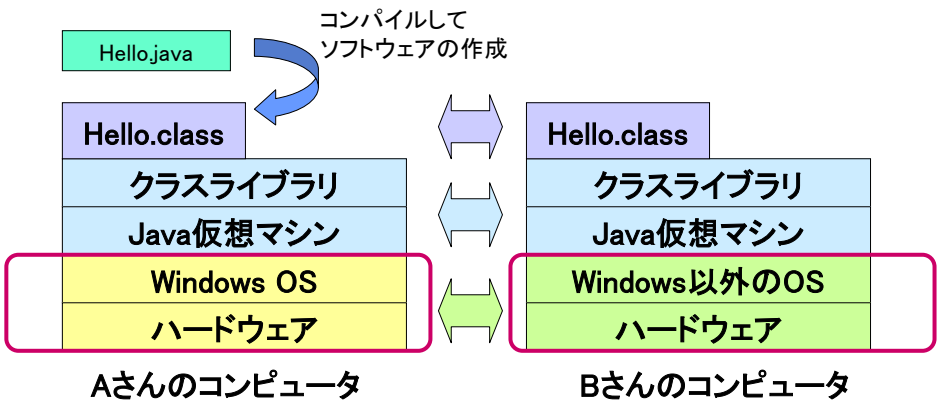
- JavaコンパイラとJavaインタプリタを利用して機械語に翻訳する

JavaとJavaScript

Javaの記述方法と若干似ている箇所があるけど、全くの別モノです

- Java……プログラミング言語の一種
  - サーブレット(**Java Servlet**)
    Javaを用いて、ウェブページのためのHTML文書などを動的に生成するサーバ上で動くプログラム、またはその仕様
  - 当初はクライアント側でのアプリケーション作成だったが、このServlet登場以降、Javaがサーバアプリケーションとして注目される
    サーバサイドJavaと呼ばれている
- JavaScript
  - スクリプト言語
  - 主にWebスクリプトとして利用される
  - Netscape社がWebスクリプト言語として開発したのが名前の由来
  - Javaとは全くの互換性がないので注意すること

Java仮想マシンによってOS非依存のソフトウェアの作成が可能(移植が簡単)



Javaを用いたプログラミングの流れ

1. ソースコード(プログラム) ～～javaの作成

2. Javaコンパイラでバイトコードを作成
   - Javaコンパイラを用いて～～javaを翻訳しバイトコード～～.classを作成

   Javaコンパイラ
   人間が書いたプログラムを、バイトコードに翻訳するソフトウェア

3. Javaインタプリタを用いて実行
   - Javaインタプリタを用いて、バイトコード～～.classファイルを実行

   Javaインタプリタ
   バイトコードを機械語に翻訳しながら実行させていくソフトウェア

オブジェクト指向

モノのあり方に着目して、現実の世界をモデル化する考え方

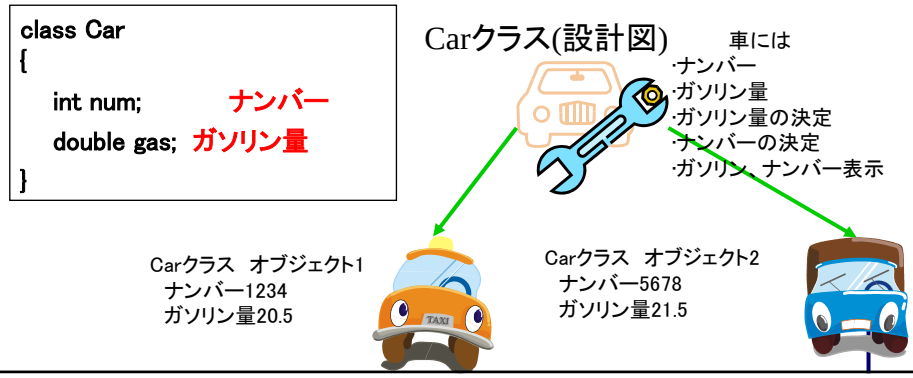
- ソフトウェア開発などで，操作の対象となるものを重視した考え方
  - 車::::
    エンジン、ハンドル、タイヤなどの部品(フィールド)を
    どのように利用するか(メソッド)を取りまとめた設計書

- オブジェクト指向の考え方のポイント
  - クラス、オブジェクト(インスタンス)
  - コンストラクタ、オーバーロード
  - 継承
  - 修飾子
- はじめにクラスを設計して、メモリ上にオブジェクトを設定する
  - クラス・・・・・・・・オブジェクトの設計図
  - オブジェクト・・・設計されたクラスに基づいて作成された実質上のデータ

クラスを利用するということは

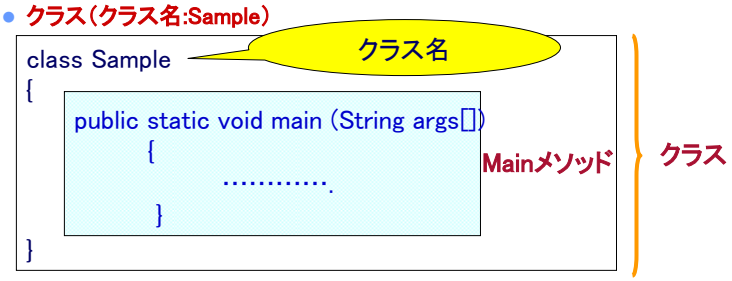
- オブジェクトの作成
  - 実際に一つのモノを作る
- オブジェクトまたはインスタンス
  - コード上で作成されるモノ一つ一つのこと
  - どのような性質をもっているかをクラスとして設計する

```
class Car
{
    int num;          ナンバー
    double gas;  ガソリン量
}
```



クラスについて

- クラス
  - Javaコードは「class」が先頭についたブロックで成立する。
    - このブロックを「クラス」という
    - 「class」の次に書かれる文字列を「クラス名」と呼ぶ
- Javaのコードには最低1つ以上のクラスが存在
  - クラス名は自由でよい（できるだけわかりやすい名前にすること）
    - クラス（クラス名:Sample）



オブジェクトの作成

- クラスを宣言する(declaration)
  - モノの性質や機能をまとめたクラスを記述すること

```
class クラス名
{
    型名 フィールド名;
     ・・・・・・・・・・・
    戻り値の型 メソッド名(引数リスト)
    {
        文1;
        ・・・・・;
        return 式;
    }
   ・・・・・・・・・・・;
}
```

フィールド
クラスの「性質」を示す
コード上では変数を用いて示す

メソッド
クラスの機能を示す

フィールドとメソッドをあわせてメンバという

メンバを用いて処理することをアクセスという

- クラスを利用する

実際に作成したクラスをどのように取り扱うか

```
class Car
{
    int     num;
    double  gus;
    String  color;

    void setCar(int n, double g, String c){
            num    = n;
            double = g;
            color  = c;
    }
    void ShowInfo(){
        System.out.println("車の番号は"+ num+"です");
        System.out.println("車のガソリンは"+ gus+"です");
        System.out.println("車の色は"+ color+"です");
    }
}
```

フィールド
クラスの「性質」を示す
コード上では変数を用いて示す

メソッド
クラスの機能を示す

アクセスの際
.(ドット)を「の」に置き換える

```
class Sample{
        public static void main(String args[]) {
                Car car1 = new Car();
                car1.setCar(1234, 20.4, "blue");
                car1.ShowInfo();
        }
}
```

Car クラスを用いた
新しいCarオブジェクトcar1の作成

car1のメソッドsetCarを用いて
各フィールドの設定
(各フィールドとsetCarのメソッドへアクセス)

car1のメソッドShowInfoメソッドへアクセス

2つ以上のオブジェクトを作成する

- オブジェクトはいくつでも作成することが可能

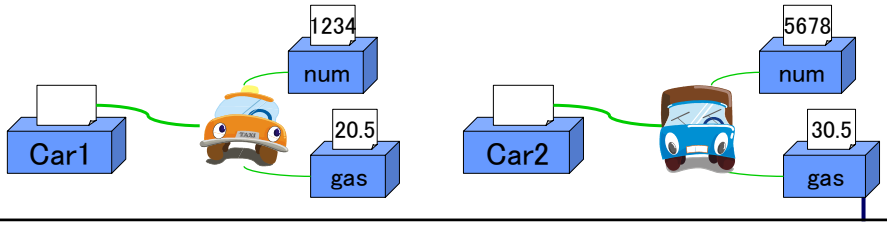
```
Car car1 ;
car1 = new Car;
car1.num = 1234;
car1.gas  =  20.5;

Car car2 ;
car2 = new Car;
car2.num = 5678;
car2.gas  =  30.5;
```

1つ目のオブジェクトの作成

1台目の車のナンバーとガソリンの代入

2つ目のオブジェクトの作成

2台目の車のナンバーとガソリンの代入



オーバーロードの仕組みを知る

メソッドのオーバーロード

同じ名前の複数のメソッドを同じクラス内に定義しておくことができる

```
public void setCar(int n)
public void setCar(double g)
public void setCar(int n, double g)
```

メソッドをオーバーロードするときには
各メソッドの引数の型・個数が異なるようにしなければならない。

引数の型・個数が異なっていれば、
同じ名前を持つメソッドだとしてもそれぞれ違うメソッドとして扱うことができる。

カプセル化の仕組みを知る（修飾子）

クラスを設計する人が、メンバを適切にprivateメンバとpublicメンバに分類しておけば、あとから他の人間がそのクラスを利用したときに、誤りのおきにくいプログラムを作成できる。

カプセル化

クラスの中のデータ（フィールド）と機能（メソッド）をひとまとめにし、保護したいメンバにprivateをつけて、勝手にアクセスできなくする機能

フィールド → privateメンバ
メソッド → publicメンバ

このような指定のカプセル化がよく行われている

メンバへのアクセスを制限する

クラスの外から勝手にアクセスできないようなメンバとしておく

privateメンバ

```
class Car
{
    private int num;
    private double gas;
    ・・・
}
```

privateメンバにすると、クラスの外から勝手にアクセスできなくなる。

```
class Sample1
{
    public static void main(String args[])
    {
        Car car1 = new Car();

        car1.num = 1234;
        car1.gas = 10.0;

        car1.show();
    }
}
```

private はフィールドでよく用いられる

publicメンバを作る

- public メンバはクラスの外からアクセスできる

```
class Car
{
    public void setNumGas(int n, double g)
    {
            ・・・・・・・・・・・・・
     }
    public void show( )
    {
            ・・・ ・・・ ・・・ ・・・ ・・・
     }
}
```

```
class Sample1
{
    public static void main(・・・・・
    {
        ・・・・・・・・・
        car1.setNumgas(1234, 20.5;
        car1.show();
    }
}
```

修飾子：クラスのメンバにアクセス制限を与えるもの

public ： クラスの外からアクセスできる
private： クラスの外からアクセスできない

コンストラクタの役割を知る

コンストラクタは、

そのクラスのオブジェクトが作成されたときに、
定義しておいたコンストラクタ内の処理が自動的に行われる。

メソッドと違って、

コンストラクタを自由に呼び出すことはできない。

コンストラクタは

オブジェクトのメンバに自動的に初期値を設定する

などの役割を書いておくのが普通。

コンストラクタの基本

```
class Car{

    private int num;
    private double gas;

    public Car(){
        num = 0;
        gas = 0.0;
        System.out.println("車を作成しました。");
    }

    public void show(){
        System.out.println("車のナンバーは" + num + "です。");
        System.out.println("ガソリン量は" + gas + "です。");
    }
}
class Sample4{

    public static void main(String args[]){

        Car car1 = new Car();

        car1.show();
    }
}
```

コンストラクタ

メソッド

実行結果

```
車を作成しました。
車のナンバーは0です。
ガソリン量は0.0です。
```

コンストラクタのオーバーロード

コンストラクタでも、メソッドと同じように、引数の数・型が異なっていれば同じようにオーバーロードすることができる。

複数のコンストラクタを定義する ことができる。

コンストラクタのオーバーロードという。

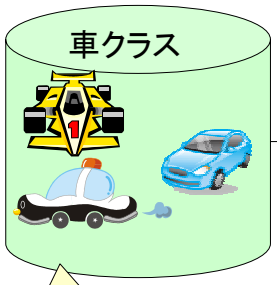
```
public Car(){ //引数なしのコンストラクタ
    num = 0;
    gas = 0.0;
    System.out.println("車を作成しました。");
}
public Car(int n, double g){ //引数を2つ持つコンストラクタ
    num = n;
    gas = g;
    System.out.println("ナンバー" + num + "ガソリン量" + gas + "の車を作成しました。");
}
```

継承とは
あるクラスの持つ機能をそっくり引き継いで、
さらに機能を付け加えた新しいクラスを作る機能

車のクラスを引き継いで、レーシングカークラス、乗用車クラスを作成

基本的なクラス
（スーパークラス）

応用的なクラス
（サブクラス）



レーシングカークラス
レースコースで走る
ピットでガソリンを入れる

乗用車クラス
公道で走る
ガソリンスタンドでガソリンを入れる

道路を走る
ガソリンを入れる

継承

```
//kurumaクラス
class Car
{
   protected int num;
   protected double gas;

   public Car()
   {
      num = 0;
      gas = 0.0;
      System.out.println("車を作成しました。");
   }
}
//レーシングカークラス(車クラスを継承したクラス)
class RacingCar extends Car
{
   private int course;

   public RacingCar()
   {
      course = 0;
      System.out.println("レーシングカーを作成しました。");
   }
}
```

extends の後ろのクラスを継承している

車クラスを更に拡張した部分

```
class Sample9
{
   public static void main(String args[])
   {
      Car cars[];
      cars = new Car[2];

      cars[0] = new Car();
      cars[1] = new RacingCar();

      for(int i=0; i<cars.length; i++){
         class cl = cars[i].getClass();
         System.out.println((i+1) + "番目の
            オブジェクトのクラスは" + cl + "です。");
      }
   }
}
```

何のクラスで定義されているかを調べるメソッド

抽象クラス

共通の機能を表現し、個々が持つ独自の機能はそれぞれのサブクラス
で実装したい場合に使用

抽象クラスのルール

- 抽象クラスのオブジェクトを生成できない
- 抽象メソッドがあるクラスは必ず抽象クラスとして宣言
- 抽象メソッドがない、抽象クラスを宣言することもできる

抽象クラス

抽象クラス

```
//のりものクラス
abstract class Vehicle
{
   protected int speed;
   public void setSpeed(int s)
   {
     speed = s;
     System.out.println("速度を"
                    + speed + "にしました。");
   }
   abstract void show();
}
```

抽象クラスの定義の修飾子は
abstract

mainメソッドを
含むクラス

```
class Sample1
{
   public static void main(String args[])
   {
      Vehicle vc[];
      vc = new Vehicle[2];

      vc[0] = new Car(1234, 20.5);
      vc[0].setSpeed(60);

      vc[1] = new Plane(232);
      vc[1].setSpeed(500);

      for(int i=0; i<vc.length; i++){
        vc[i].show();
      }
   }
}
```

抽象クラスを継承したクラス

```
//車クラス
class Car extends Vehicle
{
   private int num;
   private double gas;

   public Car(int n, double g)
   {
      num = n;
      gas = g;
      System.out.println("ナンバー" + num +
                         "ガソリン量" + gas+
                         "の車を作成しました。");
   }
   public void show()
   {
      System.out.println("車のナンバーは"
                              + num + "です。");
      System.out.println("ガソリン量は"
                               + gas + "です。");
      System.out.println("速度は"
                             + speed + "です。");
   }
}
```

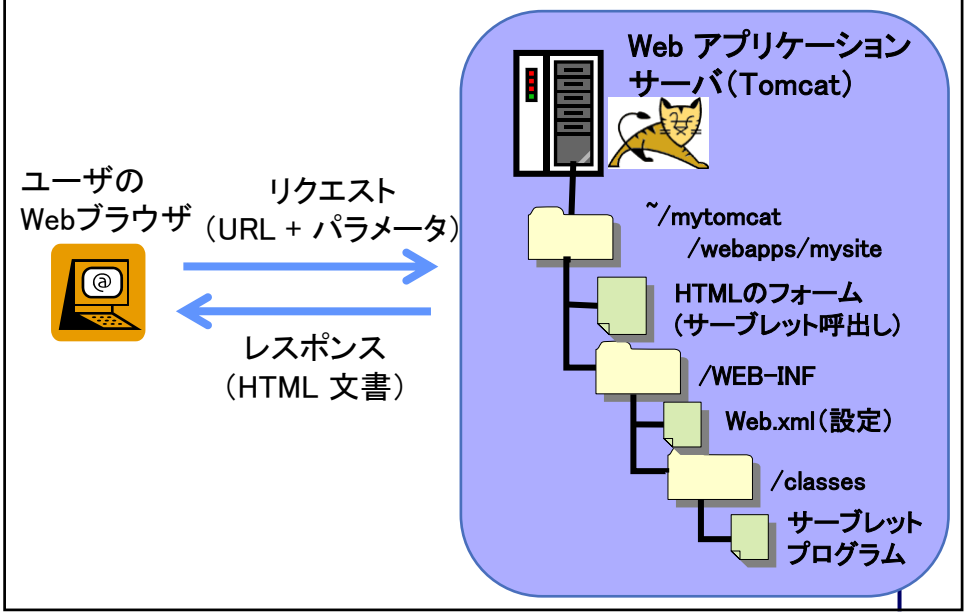
```
//飛行機クラス
class Plane extends Vehicle
{
   private int flight;

   public Plane(int f)
   {
      flight = f;
      System.out.println("便" + flight +
                         "の飛行機を作成しました。");
   }
   public void show()
   {
      System.out.println("飛行機の便は"
                          + flight + "です。");
      System.out.println("速度は"
                             + speed + "です。");
   }
}
```

Java サーブレットとは

- Java言語で作成された，Web上で動作するプログラムまたはその仕様
- クライアント(Webブラウザ)からWebサーバ上に要求があると，要求に応じて動的にHTML文章を作成し，クライアントに送信する

- CGIとサーブレットの違い
  - CGI:
    - **クライアントのリクエストの度に新しいプロセスを起動**
    - **サーバとの接続の度に毎回セッションを張りなおす必要がある**
  - **サーブレット:**
    - **一度呼び出されるとそのままメモリに常駐するので処理が高速**
    - **サーバ側でデータを並列に送信しているので，複数のユーザ間でデータの共有が可能**

サーブレットの構成



Tomcatの初期設定(資料p.15)

- 以下のスクリプトを実行

/kyozai/amaeda/2011mp1/scripts/tomcatSetup.sh

Tomcatの起動と終了

- 起動
  - ホームディレクトリに移動してから，起動スクリプトを実行

    cd

    $CATALINA_HOME/bin/startup.sh
- 終了
  - 停止スクリプトを実行

    $CATALINA_HOME/bin/shutdown.sh

（注）サーブレットのクラスファイルを更新するたびに
Tomcatの再起動が必要

web.xml

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<web-app xmlns="http://java.sun.com/xml/ns/javaee"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://java.sun.com/xml/ns/javaee
http://java.sun.com/xml/ns/javaee/web-app_2_5.xsd" version="2.5">
<servlet>
        <servlet-name>Myserv</servlet-name>
        <servlet-class>Myserv</servlet-class>
</servlet>
<servlet-mapping>
        <servlet-name>Myserv</servlet-name>
        <url-pattern>/Myserv</url-pattern>
</servlet-mapping>
</web-app>
```

サーブレットの名前

クラスファイル名（xxx.class）

url上の名前

アクセスする際はこの名前を入力
「/」から始めることに注意