

## Java オブジェクト指向

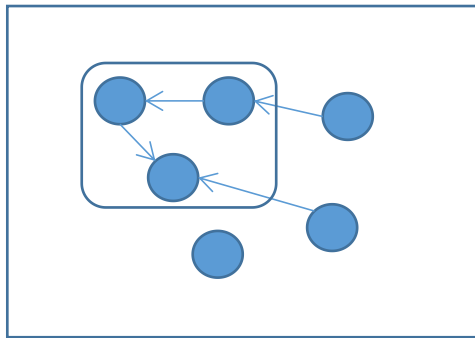
→ひと、もの、概念

これを部品化する

=クラス (=オブジェクト)

---

- ・インスタンス化
  - ・コンストラクタ
  - ・カプセル化
  - ・継承
  - ・実装
- 



オブジェクト指向をつかって、  
「向きを決めてあげる  
指示してあげる」

---

## <コマンドプロンプト>

コマンドプロンプトを開いてみましょう。

- ①Windows ボタン+R ボタン
- ②「cmd」と入力して実行する

---

## <コマンドプロンプトを使った Java プログラミング>

(演習)

以下の①～⑤をもとにプログラミングをしてみましょう。

---

(備考) notepad . . . メモ帳を開くコマンド

---

- ①コマンドプロンプトからメモ帳を開いてプログラミング

notepad HelloWorld.java

※メモ帳にプログラムを書く

---

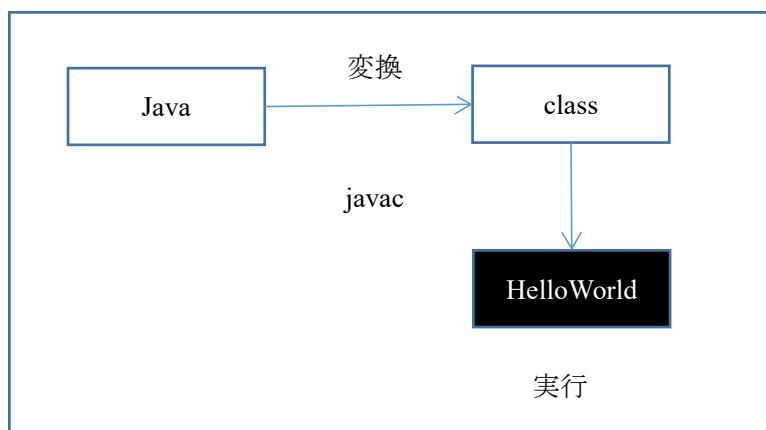
```
public class HelloWorld{  
    public static void main(String[] args){  
        System.out.println("Hello World");  
    }  
}
```

---

※終わったら保存 (Ctrl+S) する

- ②class ファイルを作成する

javac HelloWorld.java



③作成したプログラムを実行する

`java HelloWorld`

---

(備考) `del` . . . ファイルを削除するコマンド

---

④作成したファイルを一旦削除する

`del HelloWorld.java`

---

(備考) `dir` . . . ファイルやフォルダを一覧表示するコマンド

---

⑤ファイルやフォルダを一覧表示する

`dir`

---

(演習)

先ほどの①～⑤を5回繰り返しましょう。

---

(演習)

HelloWorld プログラム

以下をプログラミングして実行してみましょう。

```
String s="Hello";
```

```
System.out.println(s);
```

実行できたら、①～⑤の要領で5回繰り返しましょう。

---

(演習)

HelloWorld プログラム

以下をプログラミングして実行してみましょう。

```
String t="This is a pen";
```

```
System.out.println(t);
```

実行できたら、①～⑤の要領で5回繰り返しましょう。

---

(演習)

HelloWorld プログラム

以下をプログラミングして実行してみましょう。

```
for(int i = 0; i <=10; i++){
```

```
    System.out.println(i);
```

```
}
```

実行できたら、①～⑤の要領で5回繰り返しましょう。

---

## <インスタンス化>

### 1. 通常の書き方

```
String s = "Hello";  
System.out.println(s);
```

---

#### (演習)

HelloWorld プログラム

1. 通常の書き方を使って実行してみましょう。

実行できたら、①～⑤の要領で5回繰り返しましょう。

---

### 2. インスタンス化を使った書き方

```
String s = new String("Hello");  
System.out.println(s);
```

---

#### (演習)

HelloWorld プログラム

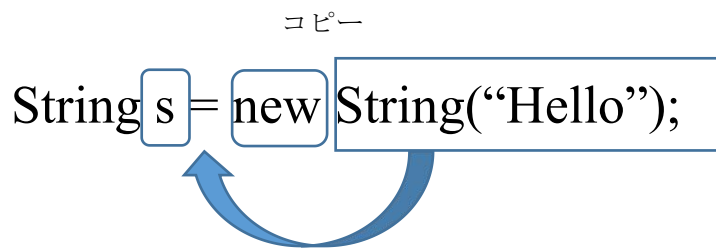
2. インスタンス化を使った書き方を使って実行してみましょう。

実行できたら、①～⑤の要領で5回繰り返しましょう。

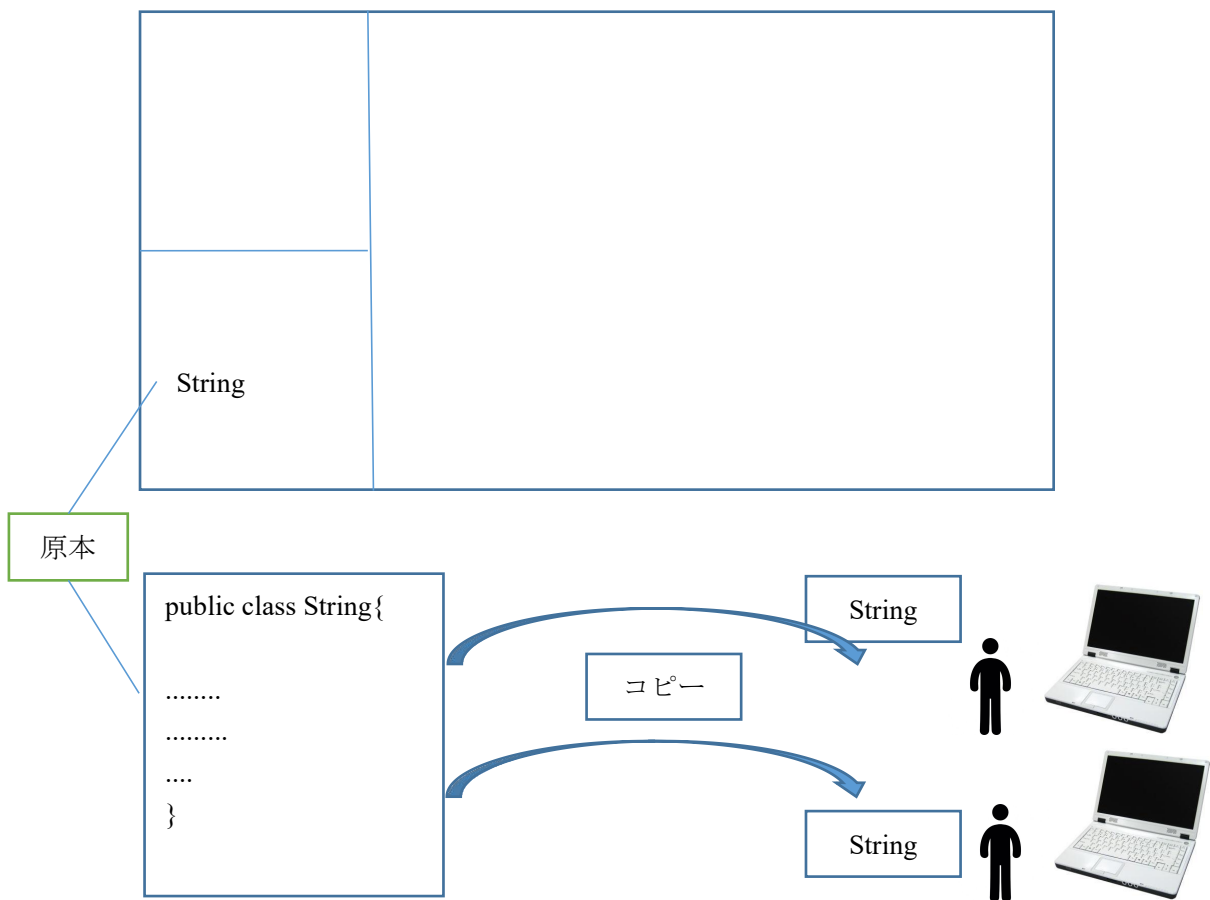
---

コピー

```
String s = new String("Hello");
```



「jdk 1.8 api」で検索する  
API ドキュメント - Overview (Java Platform SE 8 )  
<https://docs.oracle.com/javase/jp/8/docs/api/>



<インスタンス化を使った String の書き方>

```
String s = new String();  
s=" Hello";  
System.out.println(s);
```

---

(演習)

HelloWorld プログラム

上記をプログラミングして実行してみましょう。

実行できたら、①～⑤の要領で5回繰り返しましょう。

---

---

<Eclipse にフォルダを作成して、実際にプログラムする>  
ここから統合開発環境 Eclipse を使ってプログラミングします。

---

## ***HelloWorld*** プロジェクト

---

右クリック→新規→Java プロジェクト→ここでアプリケーションの名前を決定  
「HelloWorld」  
次へ→完了

```
HelloWorld
| _ src
```

右クリック→新規→クラス  
public static void main(String[] args)... にチェックをつける  
コメントの生成 にチェックをつける  
完了

<インスタンス化の練習>

---

### 1. 通常の手書き

```
String s = "Hello";
System.out.println(s);
```

---

(演習)

main メソッドに上記をプログラミングしてみましょう。  
プログラミングできたら、実行してみましょう。

---

<Eclipse で作成した Java プロジェクトの削除方法>

1. Java プロジェクトを右クリックして削除ボタン
2. ディスク上からプロジェクト・コンテンツを削除にチェック
3. OK ボタン

---

(演習)

上記の方法で Java プロジェクトを削除してみましょう。  
削除できたら、再度 Java プロジェクトを作成して3回繰り返しましょう。

---

### 2. インスタンス化を使った書き方

```
String s = new String("Hello");
System.out.println(s);
```

---

(演習)

**main** メソッドに上記をプログラミングして書き換えてみましょう。  
プログラミングできたら、実行してみましょう。

---

(演習)

**Java** プロジェクトを削除してみましょう。  
削除できたら、再度 **Java** プロジェクトを作成して3回繰り返しましょう。

---

<インスタンス化を使った String の書き方>

```
String s = new String();  
s=" Hello" ;  
System.out.println(s);
```

---

(演習)

**main** メソッドに上記をプログラミングして書き換えてみましょう。  
プログラミングできたら、実行してみましょう。

---

(演習)

**Java** プロジェクトを削除してみましょう。  
削除できたら、再度 **Java** プロジェクトを作成して3回繰り返しましょう。

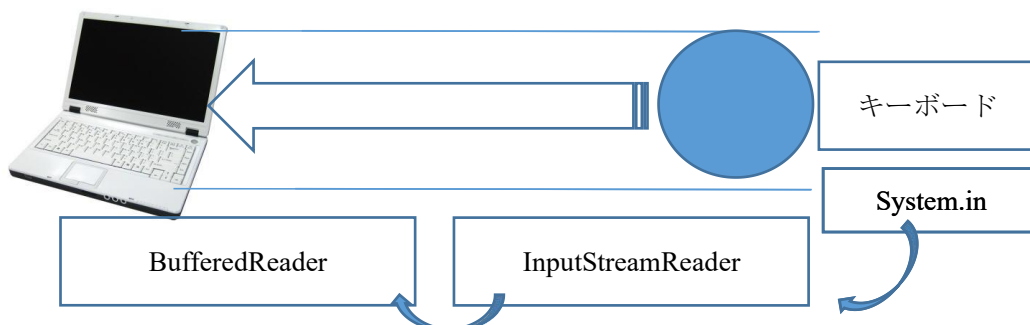
---

<文字を入力する>

```
public static void main(String[] args) throws IOException {  
    BufferedReader br = new BufferedReader(new InputStreamReader(System.in));  
    System.out.println("入力してください");  
    String t= br.readLine();  
    System.out.println(t);  
}  
}
```

追加で入力

---





(演習)

**main** メソッドに上記をプログラミングして書き換えてみましょう。  
プログラミングできたら、実行してみましょう。

---

(演習)

**Java** プロジェクトを削除してみましょう。  
削除できたら、再度 **Java** プロジェクトを作成して3回繰り返しましょう。

---

<文字列から数字に変換する>

```
String s = "123";  
int i = Integer.parseInt(s);
```

---

(演習)

**main** メソッドに上記をプログラミングして書き換えてみましょう。  
プログラミングできたら、実行してみましょう。

---

(演習)

**Java** プロジェクトを削除してみましょう。  
削除できたら、再度 **Java** プロジェクトを作成して3回繰り返しましょう。

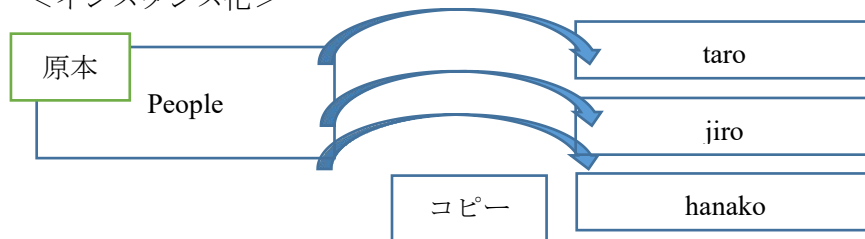
---

---

## Person プロジェクト

---

<インスタンス化>



※インスタンス化は、オブジェクト指向で最もよく使われる仕組みです。

(演習)

Java プロジェクト「Person」を作成しましょう。

「Person」クラスを作成してプログラミングしましょう。

「Test」クラスを作成してプログラミングしましょう。

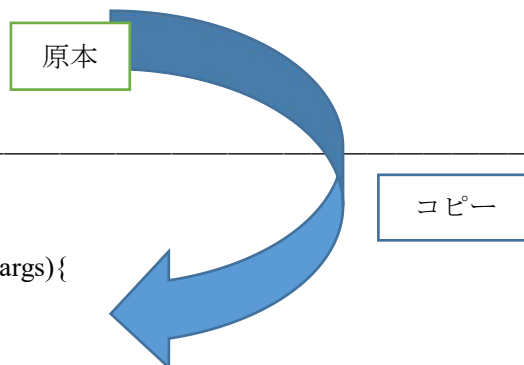
実行してみましょう。

実行できたら、一度 Java プロジェクト「Person」フォルダを削除しましょう。

5回繰り返しましょう。

```
public class Person{  
    public String name = null;  
    public int age = 0;  
}
```

```
public class Test{  
    public static void main(String[] args){  
        Person taro = new Person();  
        taro.name="山田太郎";  
        taro.age=20;  
        System.out.println(taro.name);  
        System.out.println(taro.age);  
    }  
}
```



※ Java はかならず main メソッドからスタートする。

---

(演習)

インスタンス化を使って、

木村次郎、18

鈴木花子、16

(自身のお名前)、(自身のご年齢)

を **Test** クラスにプログラミングしてみましょう。

---

(演習)

**Person** クラスに

```
public String phoneNumber = null;
```

```
public String address = null;
```

を追加してみましょう。

また、これ以外にもいくつか個人で使用するもの(役所や **Web** サイトで使用されるような個人登録情報)を追加してみましょう。

---

(演習)

さきほど **Person** クラスに追加した情報を使って、**Test** クラスにすべて表示できるようプログラミングしてみましょう。また、これを表示してみましょう。

---

(演習)

以下を参考に **Person** クラスにメソッドを追加してみましょう。

---

```
public class Person{
    public String name = null;
    public int age = 0;
    .
    .
    .
    public void talk(){
        System.out.println(this.name + “が話す”);
    }
    public void walk(){
        System.out.println(this.name + “が歩く”);
    }
    public void run(){
        System.out.println(this.name + “が走る”);
    }
}
```

```
}
```

---

(演習)

Test クラスにプログラムしたインスタンスを使って、先ほど追加したメソッドを呼び出してみましよう。

---

例)

```
taro.talk();  
taro.walk();  
taro.run();
```

---

(演習)

あたらしく Robot クラスを作成してみましよう。

この中に、name をプログラムしましよう。

また、talk()、walk()、run()メソッドをプログラムしましよう。

---

(演習)

作成した Robot クラスを使って、Test クラスでインスタンス化してみましよう。

また、インスタンスの名前は、aibo、asimo、pepper、doraemon としてプログラムしてみましよう。

完成したら、実行してみましよう。

---

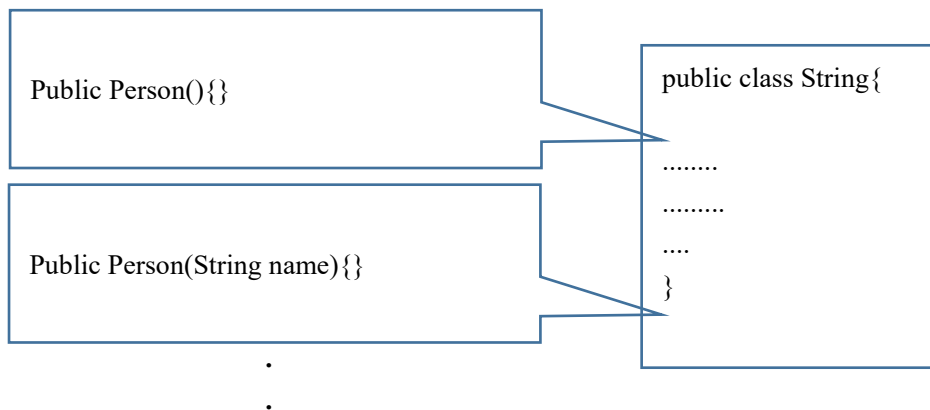
---

## Person プロジェクト

---

### <コンストラクタ>

Java は「コンストラクタ」というプログラムを呼び出して、インスタンス化します。

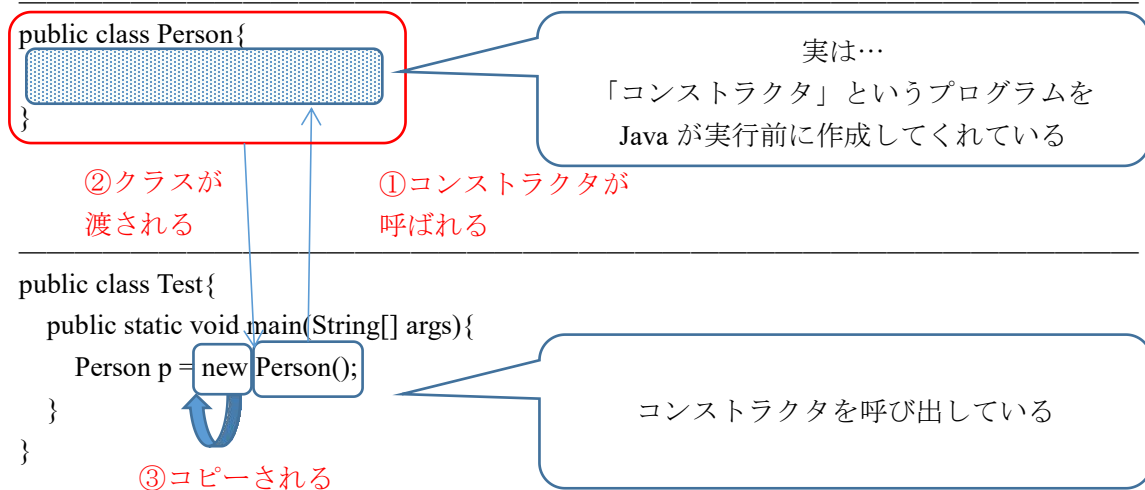


---

### (演習)

Java プロジェクト「Person」を作成しましょう。

以下の Test クラスをプログラミングしてみましょう。



(演習)

以下の Person クラスをプログラミングしてみましょう。

```
public class Person{  
    public Person(){}  
}
```

コンストラクタをプログラミング  
することも可能

```
public class Test{  
    public static void main(String[] args){  
        Person p = new Person();  
    }  
}
```

コンストラクタを呼び出している

<インスタンス化とコンストラクタ>

※ コンストラクタはいくつもの種類を作成できる  
このことを「多重定義（オーバーロード）」という

(演習)

Person クラスにコンストラクタ②を追加してみましょう。

```
public class Person{  
    public String name = null;  
    public int age = 0;
```

```
    public Person(){}  
}
```

コンストラクタ①

```
    public Person(String name, int age){  
        this.name = name;  
        this.age = age;  
    }  
}
```

コンストラクタ②  
(あたらしく追加)

※ this. をつけると、「このクラスの・・・」という意味

(演習)

Test クラスに以下のプログラムを追加してみましょう。

---

```
Person taro = new Person();
taro.name = "taro";
taro.age = 18;
System.out.println(taro.name);
System.out.println(taro.age);
```

```
Person jiro = new Person("jiro", 20);
System.out.println(jiro.name);
System.out.println(jiro.age);
```

---

(演習)

Person クラスにコンストラクタ③～⑤を追加してみましょう。

---

```
public Person() {}
```

コンストラクタ①

```
public Person(String name, int age) {
    this.name = name;
    this.age = age;
}
```

コンストラクタ②

```
public Person(String name) {
    this.name = name;
    this.age = 0;
}
```

コンストラクタ③  
(あたらしく追加)

```
public Person(int age) {
    this.name = "名前なし";
    this.age = age;
}
```

コンストラクタ④  
(あたらしく追加)

```
public Person(int age, String name) {
    this.name = name;
    this.age = age;
}
```

コンストラクタ⑤  
(あたらしく追加)

---

(演習)

Test クラスを使って以下の条件でプログラムを実行してみましょう。

コンストラクタ③を使ってインスタンス化・・・ **saburo** と **0** を表示しましょう。

コンストラクタ④を使ってインスタンス化・・・ 名前なしと **25** を表示しましょう。

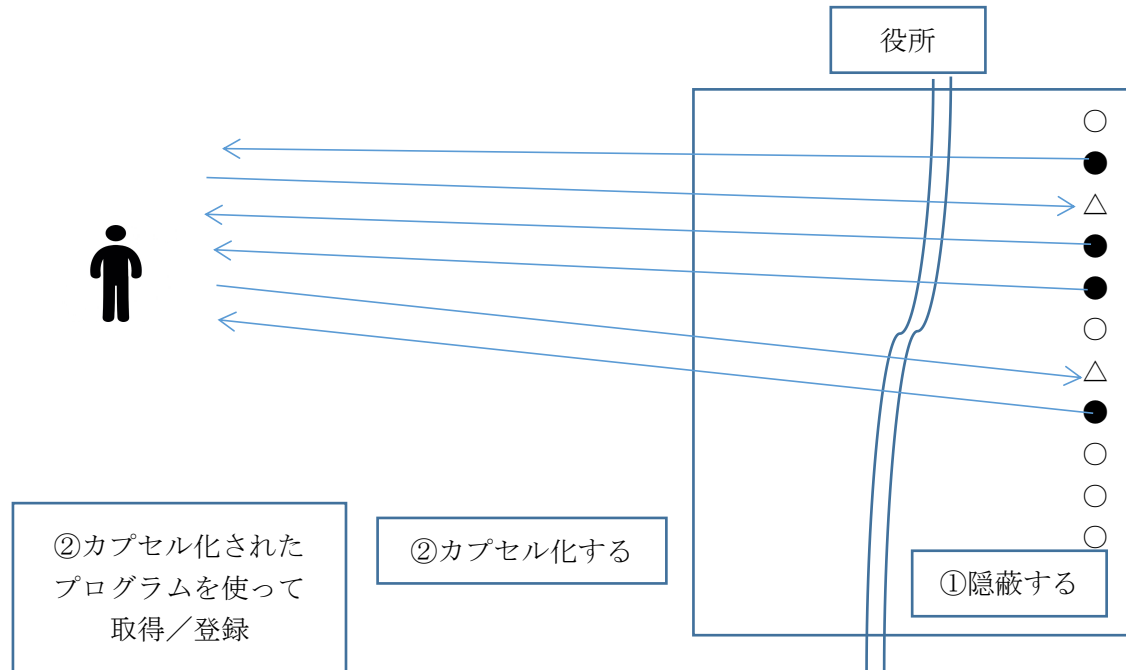
コンストラクタ⑤を使ってインスタンス化・・・ **hanako** と **17** を表示しましょう。

---



### <カプセル化>

必要な情報のみ取得したり、登録したりする為のやり方



### <隠蔽とカプセル化の書き方>

#### (①隠蔽する)

```
private String name = null;  
private int age = 0;
```

#### (②カプセル化する)

```
public String getName() {  
    return this.name;  
}
```

```
public void setName() {  
    This.name = name;  
}
```

---

## Capsule プロジェクト

---

(演習)

Java プロジェクト「Capsule」を作成しましょう。

以下の Person クラスを作成後、プログラミングしましょう。

---

```
public class Person{  
    public String name = null;  
    public int age = 0;
```

(隠蔽部分)

これを **private** に変更する。

※次の「スコープ」で解説します。

(カプセル化部分)

```
        public String getName() {  
            return this.name;  
        }  
  
        public void setName(String name) {  
            this.name = name;  
        }  
    }  
}
```

(演習)

以下の Capsule クラスを作成後、プログラミングしましょう。

---

```
public class Capsule{  
    public static void main(String[] args) {  
        Person taro = new Person(“山田太郎”, 20);  
        System.out.println(taro.name);  
    }  
}
```

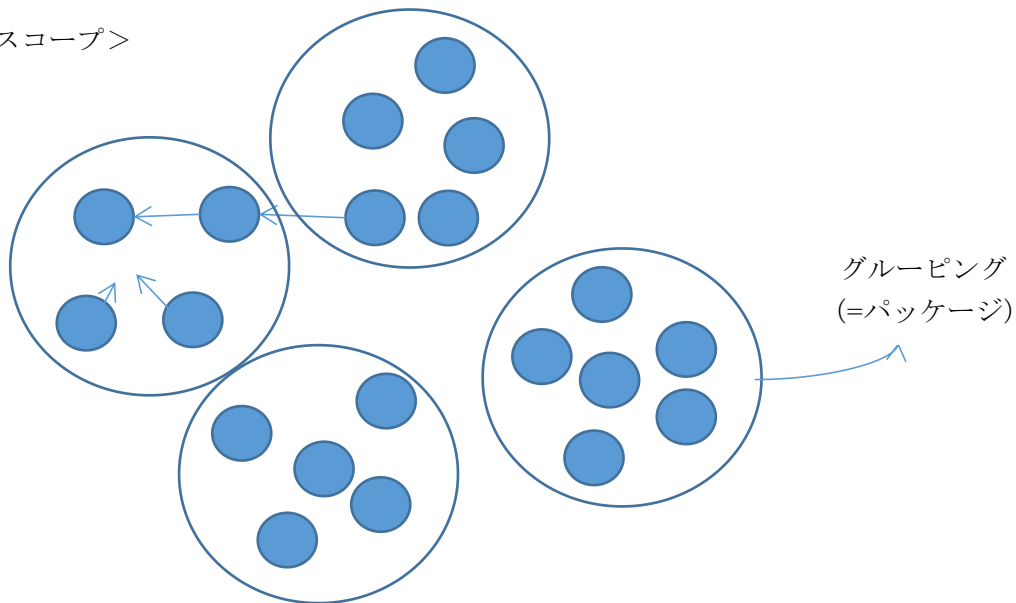
---

(演習)

Capsule プロジェクトを実行してみましょう。

---

## <スコープ>



---

## <スコープの見せ方>

※Java には 4 つの アクセス修飾子 がある

- private      - 自分しか該当のプログラムが見えない
- 何も書かない - 同じパッケージの中のプログラムは見える  
(デフォルト)
- protected   - 同じパッケージの中+特別に許されたプログラムは見える
- public        - すべてのプログラムが見える

---

## <フィールドの書き方>

アクセス修飾子 データ型 変数名 = 値/情報;

```
public class クラス名 {  
    public String name = "xxx";
```

クラスの中且つメソッド以外の場所に  
プログラミングする。  
プログラミングしたクラスの中であれば  
どこからでも使用できる。

---

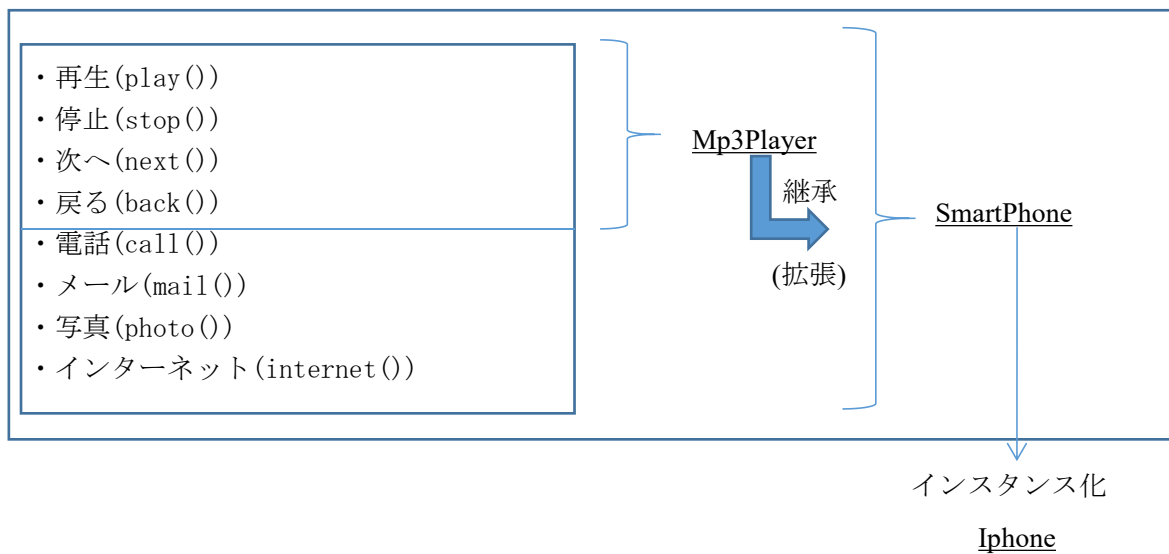
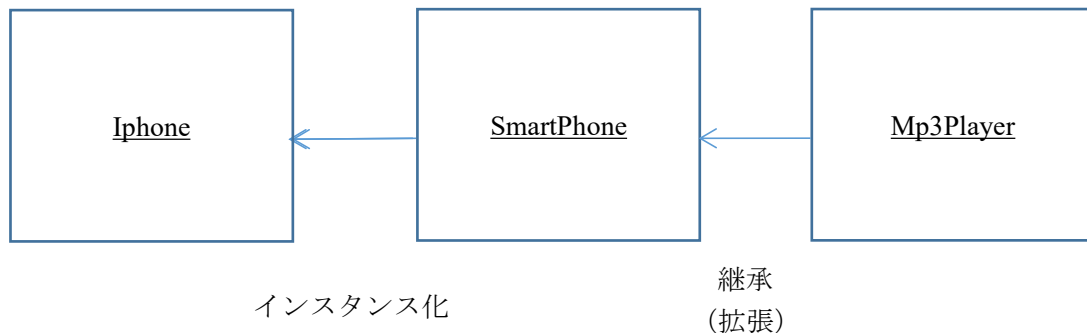
## *Iphone* プロジェクト

---

### <継承>

他のクラスがもつ機能を使って拡張する。

拡張したいプログラムは自身のクラスに追加する。



サブクラス  
(=子クラス(=this))

スーパークラス  
(=親クラス(=super))

```
public class SmartPhone extends Mp3Player{  
    .  
    .  
}
```

継承(拡張)

---

## <Iphone プログラムを作成する>

---

### (演習)

Java プロジェクト「Iphone」を作成しましょう。

以下の Mp3Player クラスを作成後、プログラミングしましょう。

---

```
public class Mp3Player{
    public void play() {
        System.out.println("再生");
    }
    public void stop() {
        System.out.println("停止");
    }
    public void next() {
        System.out.println("次へ");
    }
    public void back() {
        System.out.println("戻る");
    }
}
```

---

### (演習)

以下の SmartPhone クラスを作成後、プログラミングしましょう。

---

```
public class SmartPhone extends Mp3Player{
    public void call() {
        System.out.println("電話");
    }
    public void mail() {
        System.out.println("メール");
    }
    public void photo() {
        System.out.println("写真");
    }
    public void internet() {
        System.out.println("インターネット");
    }
}
```

---

---

(演習)

以下の Iphone クラスを作成後、プログラミングしましょう。

---

```
public class Iphone{
    public static void main(String[] args){
        SmartPhone iphone = new SmartPhone();
        iphone.play();
        iphone.stop();
        iphone.next();
        iphone.back();
        iphone.call();
        iphone.mail();
        iphone.photo();
        iphone.internet();
    }
}
```

---

(演習)

Iphone プロジェクトを実行してみましょう。

---

---

<オーバーライド>

スーパークラス（親クラス）の機能をサブクラス（子クラス）で上書きすることもできる

```
public class SmartPhone extends Mp3Player{
    public void play() {
        System.out.println("再生 (ハイRez機能) ");
    }
    public void stop() {
        System.out.println("停止 (ハイRez機能) ");
    }
    public void next() {
        System.out.println("次へ (ハイRez機能) ");
    }
    public void back() {
        System.out.println("戻る (ハイRez機能) ");
    }
}
```

オーバーライドした  
プログラム

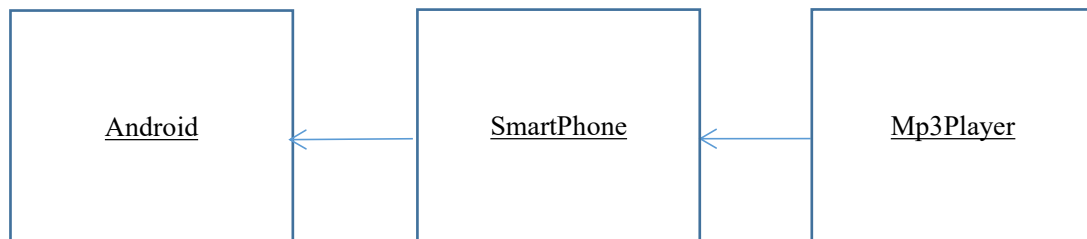
→Mp3Player の同名メソッドは無視されて、こちらが実行される。

```
}  
public void call() {  
    System.out.println("電話");  
}  
public void mail() {  
    System.out.println("メール");  
}  
public void photo() {  
    System.out.println("写真");  
}  
public void internet() {  
    System.out.println("インターネット");  
}  
}
```

---

(演習)

Iphone プログラムをもとに、Android プログラムを作ってみましょう。  
構成は以下の通りとします。



インスタンス化

継承  
(拡張)

---

## <実装>

ルールを作る

クラス (class) → 処理をプログラミング  
インターフェイス (interface) → ルールをプログラミング

---

```
public interface Mp3Player {  
    public abstract void play();  
    public abstract void stop();  
    public abstract void next();  
    public abstract void back();  
}
```

抽象メソッド

アブストラクト  
(抽象的な)

---

```
public class SmartPhone implements Mp3Player {
```

```
    public void play() {  
        System.out.println("再生");  
    }  
    public void stop() {  
        System.out.println("停止");  
    }  
    public void next() {  
        System.out.println("次へ");  
    }  
    public void back() {  
        System.out.println("戻る");  
    }  
}
```

**implements** : 実装  
ここで書いたインターフェイス  
でルール化

※必ずインターフェイスに書いた  
モノを使ってプログラムする

}



---

```
public class Iphone{  
    public static void main(String[] args) {  
        SmartPhone iphone = new SmartPhone();  
        iphone.play();  
        iphone.stop();  
        iphone.next();  
        iphone.back();  
    }  
}
```

---

ここで Iphone プロジェクトを実行してみましょう。

---

<実装の応用>

※implements の後ろにインターフェイスは複数追加可能。

例)

```
public class XXXXX implements AAAAA,BBBBB, CCCCC ..... {  
  
}
```

---

インターフェイスをさらに作成しましょう。

---

```
public interface NewFunction {  
  
    public abstract void call();  
    public abstract void mail();  
    public abstract void photo();  
    public abstract void internet();  
  
}
```

抽象メソッド

---

継承されるクラスを作成する。

---

```
public class Phone {  
  
    public void play() {  
        System.out.println("再生");  
    }  
    public void stop() {  
        System.out.println("停止");  
    }  
    public void next() {  
        System.out.println("次へ");  
    }  
}
```

```
public void back() {  
    System.out.println("戻る");  
}  
public void call() {  
    System.out.println("電話");  
}  
public void mail() {  
    System.out.println("メール");  
}  
public void photo() {  
    System.out.println("写真");  
}  
public void internet() {  
    System.out.println("インターネット");  
}  
}
```

---

継承と実装を使ってプログラミング。

---

```
public class SmartPhone extends Phone implements Mp3Player, NewFunction{  
}
```

---

SmartPhone をインスタンス化する。

---

```
public class Iphone{  
    public static void main(String[] args) {  
        SmartPhone iphone = new SmartPhone();  
        iphone.play();  
        iphone.stop();  
        iphone.next();  
        iphone.back();  
        iphone.call();  
        iphone.mail();  
        iphone.photo();  
        iphone.internet();  
    }  
}
```

---

## <継承と実装の組み合わせ>

継承と実装は同時に使用することができる。

```
public interface Mp3Player {  
    public abstract void play();  
    public abstract void stop();  
    public abstract void next();  
    public abstract void back();  
}
```

抽象メソッド

アブストラクト  
(抽象的な)

```
public class SmartPhone implements Mp3Player {
```

```
    public void play() {  
        System.out.println("再生");  
    }  
    public void stop() {  
        System.out.println("停止");  
    }  
    public void next() {  
        System.out.println("次へ");  
    }  
    public void back() {  
        System.out.println("戻る");  
    }  
}
```

**implements** : 実装  
ここで書いたインターフェース  
でルール化

※必ずインターフェースに書いた  
モノを使ってプログラムする

```
}
```

---

```
public class Iphone{
    public static void main(String[] args) {
        SmartPhone iphone = new SmartPhone();
        iphone.play();
        iphone.stop();
        iphone.next();
        iphone.back();
    }
}
```

---

※implements の後ろにインターフェイスは複数追加可能。

例)

```
public class XXXXX implements AAAAA, BBBBB, CCCCC ..... {

}
```

---

インターフェイスをさらに作成する。

---

```
public interface NewFunction {

    public abstract void call();
    public abstract void mail();
    public abstract void photo();
    public abstract void internet();

}
```

抽象メソッド

---

継承されるクラスを作成する。

---

```
public class Phone {

    public void play() {
        System.out.println("再生");
    }
    public void stop() {
        System.out.println("停止");
    } public void next() {
        System.out.println("次へ");
    }
    public void back() {
        System.out.println("戻る");
    }
    public void call() {
        System.out.println("電話");
    }
    public void mail() {
        System.out.println("メール");
    }
    public void photo() {
        System.out.println("写真");
    }
    public void internet() {
        System.out.println("インターネット");
    }

}
```

---

継承と実装を使ってプログラミング。

---

```
public class SmartPhone extends Phone implements Mp3Player, NewFunction{
}
```

---

SmartPhone をインスタンス化する。

---

```
public class Iphone{  
    public static void main(String[] args) {  
        SmartPhone iphone = new SmartPhone();  
        iphone.play();  
        iphone.stop();  
        iphone.next();  
        iphone.back();  
        iphone.call();  
        iphone.mail();  
        iphone.photo();  
        iphone.internet();  
    }  
}
```

---

---

Iphone プロジェクトを実行してみましょう。

---