Woodland Academy

2.3カプセル化

- Java パッケージ
- カプセル化
- カプセル化の文法

Shape Your Future

目次

Java パッケージ

- **パッケージ**[Package]とは、**複数のクラスをグループ化**するためのものです。実は**すべてのJavaプログラムは必ず何かしらのパッケージに属しています。**逆に**パッケージで管理されていないファイルはJavaのソースコードとして認められません。**

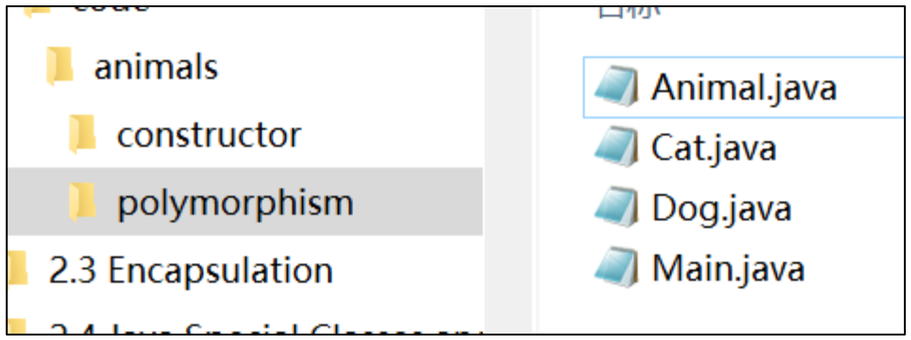
- パッケージは、コンピュータの**フォルダ構造**に似てます。異なる問題を扱うコードを違うパッケージに配置することで、コードを簡単に整理できます。

- 名前が同じなクラスを違うパッケージに配置することもできて（例えば、java.awt.Window と my.house.Window）、衝突を回避できます。（即ち、**名前空間**[Namespace]を提供できます。）

Woodland Academy

パッケージの宣言

- コードがどのパッケージに属するかを宣言するには、**２つ**のステップが必要：
  1. パッケージと同じ名前の**フォルダ**にコードを配置：



ここで、「Animal.java」は「animals/polymorphism」フォルダに置かれて、対応するパッケージは「animals.polymorphism」になっています。

2. コードの最初にパッケージ名を **package** キーワードで宣言：

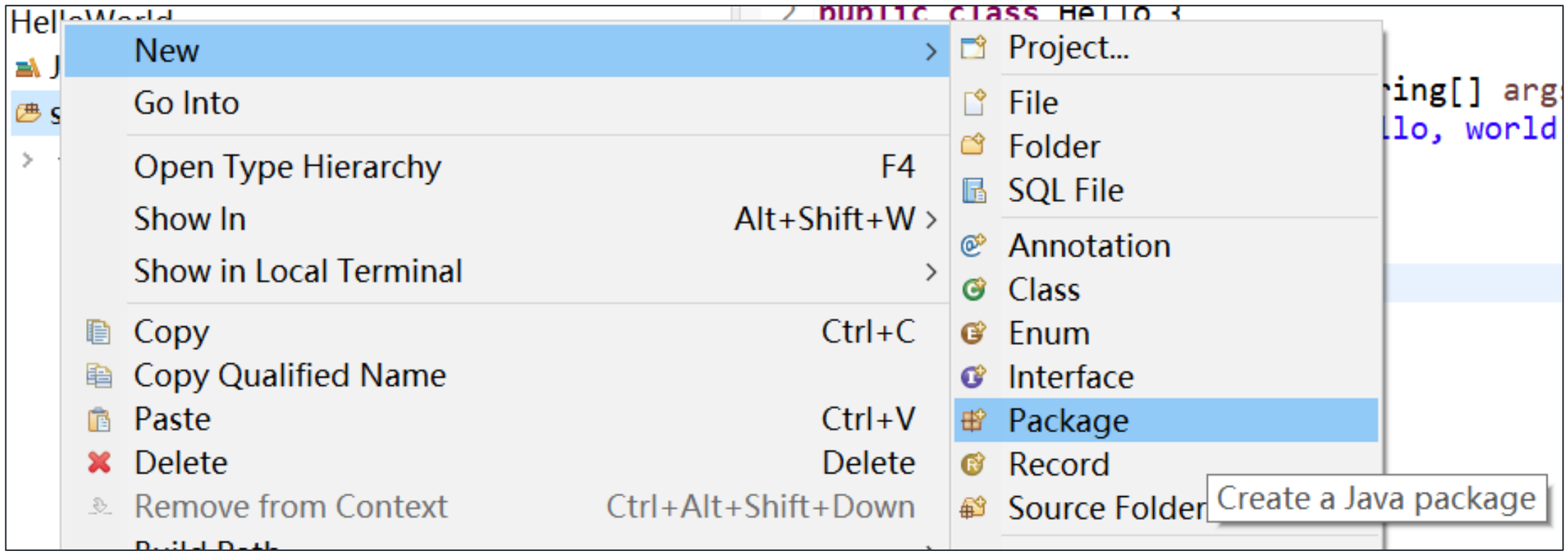
```
package animals.polymorphism;
```

次へ

前へ

- 面倒くさいと感じますか？Eclipse（そしてほとんどのJava IDE）では新規パッケージ機能を直接使用することができ、新しいコードファイルには自動的にパッケージ宣言が追加されます。

- src フォルダまたは既存のパッケージを右クリック → New → Package。



パッケージの使用

- 他のパッケージで定義されているクラスを利用したい場合は、まず対応するパッケージから該当するクラスを**インポート**する必要があります。クラスをインポートするには、**import** キーワードを使用し、その後にパッケージ名とクラス名を「**.**」で繋げて書きます：

```
import animals.polymorphism.Cat;
```

- また、アスタリスク「*」を使用することで、パッケージ内のすべてのクラスを一度にインポートできます：

```
import animals.polymorphism.*;
```

クラス名の衝突

- 異なるパッケージに同じ名前のクラスが 2 つもある場合は、クラスを区別するために、このように**フルパス**を記述する必要があります：

```
java.awt.Window window = new java.awt.Window( );
my.house.Window houseWindow = new my.house.Window( );
```

パッケージまとめ

«パッケージとは»

- パッケージとは**複数のクラスをグループ化**するためのものです。 実は**すべてのJavaプログラムは必ず何かしらのパッケージに属して います**。逆**にパッケージで管理されていないファイルはJavaの ソースコードとして認められません**。

- ソースコードの先頭で**パッケージ宣言**することでそのクラスが所属 するパッケージを定義することができます。なお、1つのクラスが 所属できるパッケージは1つのみです。 パッケージ宣言は以下のように書きます。

package パッケージ名；

- パッケージは**必ずパッケージ名と同じ名前のフォルダで管理**されな ければなりません。この仕様のため、**パッケージ＝フォルダ**と捉えて いただいても特に支障はありません。

- パッケージ宣言で指定するパッケージ名はCLASSPATHからの 相対パスとなります。 **CLASSPATHは環境変数の1つで、 Java実行時のアクセス先として PC上におけるJavaのクラスやパッケージの格納場所を管理**します。CLASSPATHの設定をしないと基本的にはJVMがアクセスできずに エラーとなります。

«宣言不要のパッケージ»

- **実行ファイルが存在するフォルダ（カレントディレクトリ）を自動的にパッケージとして扱われます**。これを**無名のパッケージ**といいます。

- 無名のパッケージに格納されているファイルは**すべてpackage宣言が不要**です。

«パッケージとインポートの要否»

- **同じパッケージ内であればインポート宣言なし**で他のクラスを利用 することができます。

- **パッケージ外のクラスを利用する場合はインポート宣言が必要**に なります。（APIのjava.langパッケージを除く）

Woodland Academy

Woodland Academy

Q&A

Coffee Break

パッケージの命名方法

パッケージ名は小文字の英単語で構成します。複数の単語からなる場合はアンダースコア「_」で繋げます。

ただし、パッケージの命名にはこういう広く知られているルールもある：パッケージ名の最初の数個の単語は開発会社・組織が使用する**ドメイン名**（すなわち、ウェブサイトのアドレス）の**逆順**で構成します。例えば、ホームページの URL が「home.zhangsan.com」である場合に作成した hello パッケージの名は次のようになります：

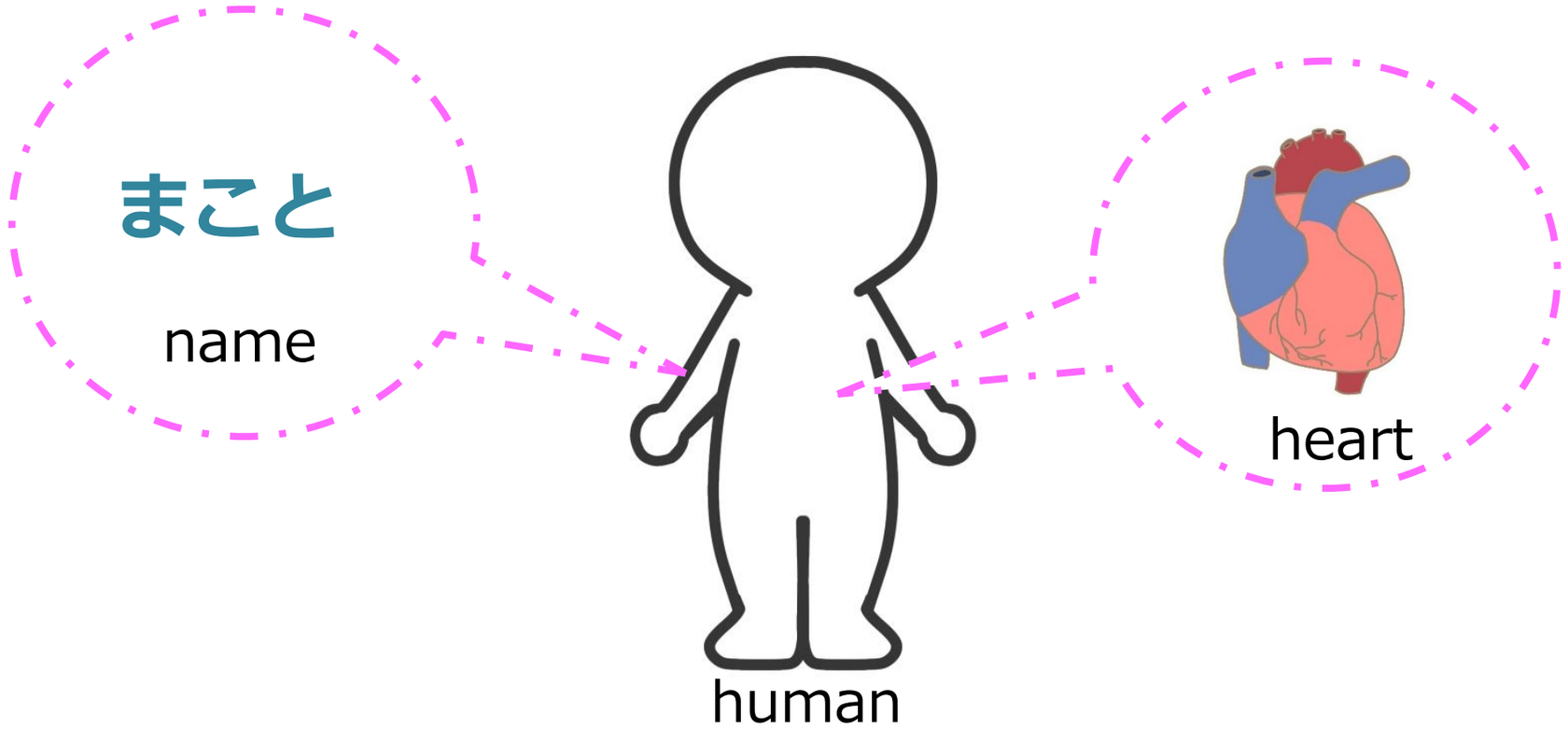
```
package com.zhangsan.hello;
```

後ほど学習する Spring Boot アーキテクチャは、「org.springframework.boot」という基本パッケージ名を使っています。Springのオフィシャルサイトの URL を推測しましょう。

目次

カプセル化の必要性

- 現実世界のモノで想像してほしいのですが、モノというのは本来外部からアクセスできる情報や機能はそこまで多くありません。
- 例えば人間だと急所である心臓という機能を肉体で隠していますし、名前も誰かに変えられるようなことはないよう法律で守られています。
- オブジェクト指向で表現するモノも同じく、フィールドやメソッドを無制限に外部に公開していると、ミスや悪意ある攻撃などによるトラブルが起きやすくなってしまいます。
- これらのトラブルを防いでくれる考え方がカプセル化となります。



カプセル化とは

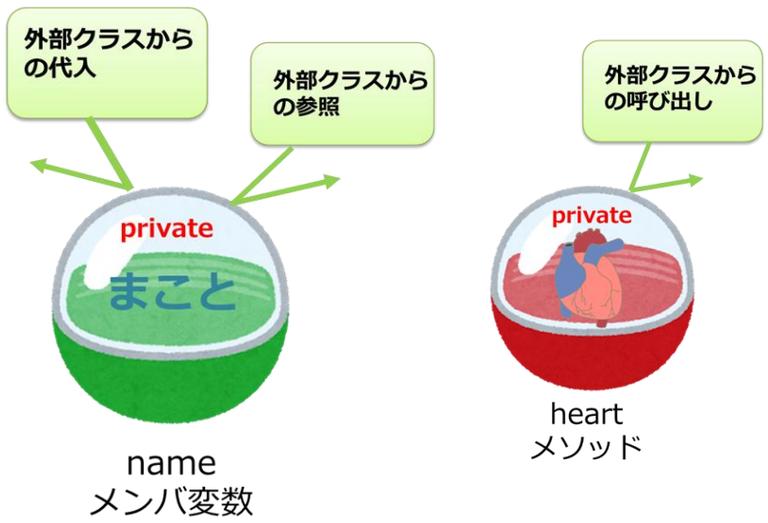
- privateやpublicといったアクセス修飾子を用い、外部クラスからのフィールドやメソッドへのアクセスを制御することをカプセル化と言います。

- カプセル化のポイントは以下の通りです。
  - ミス・悪意によるフィールドの書き換えや意図せぬ用途でメソッドが起動されるなどの外部アクセスによるトラブルを防ぐため、外部に公開する必要のないフィールドやメソッドはすべて隠す。

- - 外部にメンバ（特にフィールド）へのアクセスを許す場合も、そのメンバにアクセスするための正式な手続き方法（アクセス用メソッド）のみを公開する

目次

Java 中のカプセル化




```
public class User {
    private String username;

    private void heart() {
        System.out.println("ドキドキ");
    }
}
```



```
public class Computer {
    //使用するマウス名
    public String useMouseName;
    //使用機材名
    private String nameOfSubstrateUsed;
}
```

Woodland Academy

«アクセス制御（privateとpublic）»

- **privateは自クラス内からのアクセスのみを許す修飾子**です。カプセル化のキモはこのprivateをフィールドやメソッドにつけることで、他クラスからその存在を隠すことにあります。

- privateとは逆に**外部クラスに公開したいフィールドやメソッドにはpublic修飾子**をつけます。

- オブジェクト指向では**基本的にフィールドには**
  privateをつけます。フィールドにpublicをつけるケースは極めて稀だと思ってください。フィールドにprivateをつけると、そのフィールドは外部クラスから**変更はおろか参照すらできなくなります**。

- クラスで内部的に使うことを想定したメソッドには必ずprivateをつけましょう。外部に公開するメソッドは基本的にpublicをつけます。

やってみよう

- 今回は、銀行クラスを例にソースを書いてみよう！
  ・メンバ変数
  ユーザー名（公開する）：文字列　username
  残高(非公開)　　:整数　　balanceOfBank
  パスワード（非公開）　:整数　　password

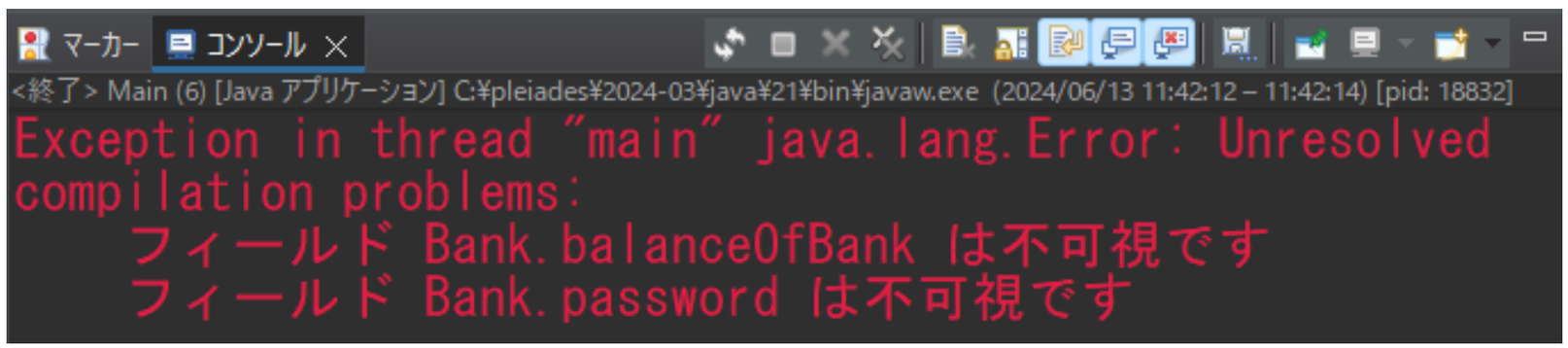
  ・上記3つの値を受けとりメンバ変数に代入できる
  コンストラクタを書いてください。

- 確認すること
  非公開のものは、呼び出すことができないことを確認してください。

解答

```java
public class Bank {
    //ユーザー名
    public String username;
    //残高
    private int balanceOfBank;
    //パスワード
    private int password;

    //コンストラクタ
    public Bank(String username, int balanceOfBank, int password) {
        this.username = username;
        this.balanceOfBank = balanceOfBank;
        this.password = password;
    }
}
```

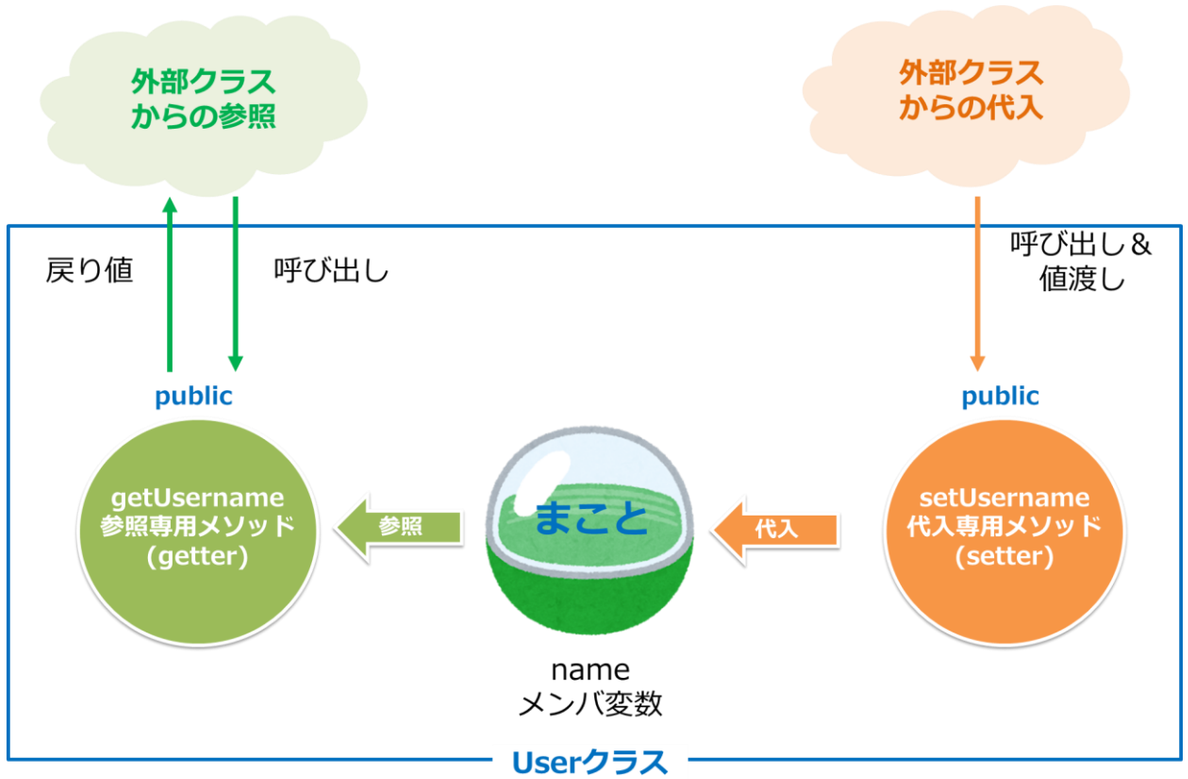
```java
public class Main {

    public static void main(String[] args) {
        Bank bank = new Bank("Alice", 1200, 1234);

        //ユーザー名を呼び出す
        System.out.println(bank.username);
        //残高を呼び出す
        System.out.println(bank.balanceOfBank);
        //パスワードを呼び出す
        System.out.println(bank.password);
    }
}
```



メンバ変数に「private」がついているものは、呼び出すことができないことは確認できたでしょうか？
次からは、どうすれば、privateの内容を呼び出せたり、編集できるかを見ていきます！

Woodland Academy

アクセス用メソッド（getterとsetter)




```java
public class User {
    private String username;

    private void heart() {
        System.out.println("ドキドキ");
    }

    //ゲッターとセッター
    public String getUsername() {
        return username;
    }

    public void setUsername(String username) {
        this.username = username;
    }
}
```

- privateをつけたフィールドには外部から参照・変更ができるような専用のメソッドを必要に応じて提供します。これをアクセス用メソッドと言い、参照用メソッドをgetter、代入用メソッドをsetterと呼びます。
- getterは引数なしで呼び出されると、対応するprivateなフィールドの値を参照し、戻り値として返すだけのメソッドです。
  以下のように記述します。
  （フィールド名を「abc」とする）

  (public) フィールドの型 getAbc() { return abc; }

- setterは引数を対応するprivateなフィールドに代入するだけのメソッドです。
  以下のように記述します。
  フィールド名を「abc」とする）

  (public) void setAbc(フィールドの型 引数名)
  { abc = 引数名 };

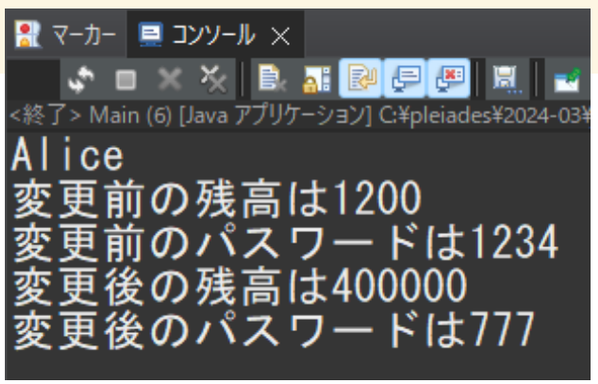
やってみよう

- P16のやってみようで書いた銀行クラスにgetterとsetterを付け足して、
  - ・passwordを前の値と異なる内容を代入して表示
  - ・balanceOfBankを前の値と異なる内容を代入して表示してください。

解答

```java
public class Bank {
    //ユーザー名
    public String username;
    //残高
    private int balanceOfBank;
    //パスワード
    private int password;

    //コンストラクタ
    public Bank(String username, int balanceOfBank, int password) {
        this.username = username;
        this.balanceOfBank = balanceOfBank;
        this.password = password;
    }

    //ゲッターとセッター
    public int getBalanceOfBank() {
        return balanceOfBank;
    }

    public void setBalanceOfBank(int balanceOfBank) {
        this.balanceOfBank = balanceOfBank;
    }

    public int getPassword() {
        return password;
    }

    public void setPassword(int password) {
        this.password = password;
    }
}
```

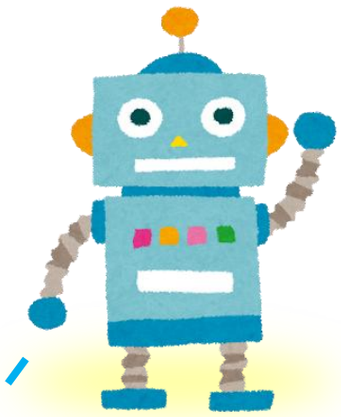
```java
public class Main {

    public static void main(String[] args) {
        Bank bank = new Bank("Alice", 1200, 1234);

        //ユーザー名を呼び出す
        System.out.println(bank.username);
        //変更前の残高を呼び出す
        System.out.println("変更前の残高は"+bank.getBalanceOfBank());
        //変更前のパスワードを呼び出す
        System.out.println("変更前のパスワードは"+bank.getPassword());

        //残高の値を変更
        bank.setBalanceOfBank(400000);
        //パスワードの値を変更
        bank.setPassword(777);

        //変更後の残高を呼び出す
        System.out.println("変更後の残高は"+bank.getBalanceOfBank());
        //変更後のパスワードを呼び出す
        System.out.println("変更後のパスワードは"+bank.getPassword());
    }
}
```



ゲッターとセッターという仲介人を介すことで、
Bankクラスにアクセスすることができれば完璧です！

Woodland Academy

mainメソッドにpublicが用いられる理由



JVM

mainメソッド起動

```
public static void main(String[] args) {}
```

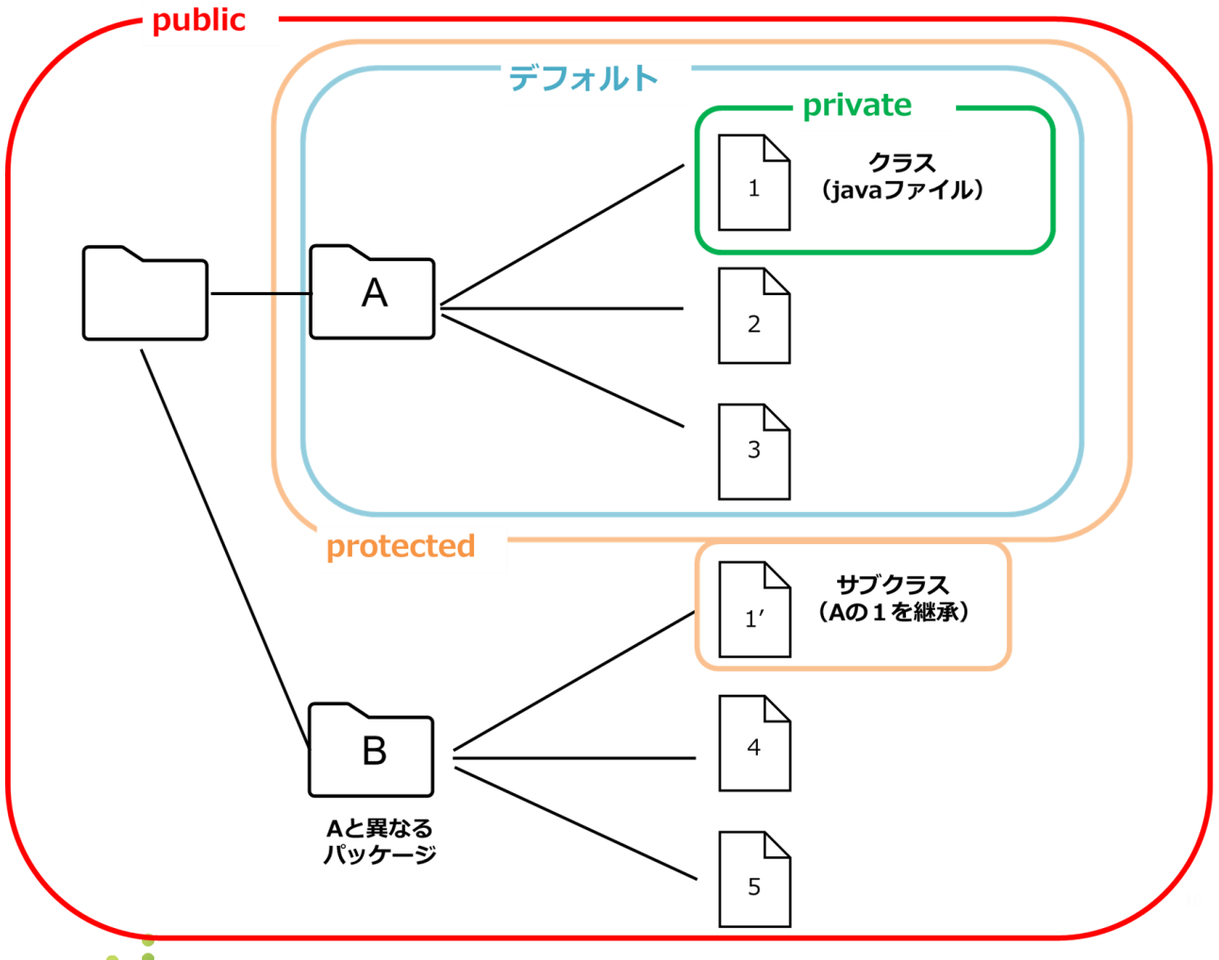
mainメソッドはJVMから呼び出されますが、JVM自体がクラスの外にあるためpublicが必要になります。

```
public class Main {

    public static void main(String[] args) {
        double result = area(12.5,2.0);
        System.out.println(result);

    }

    //面積を求めるメソッド
    public static double area(double width,double height) {
        return width * height;
    }
}
```

Woodland Academy

アクセス修飾子

アクセス修飾子	説明
public	どこからでもアクセス可能
protected	同じパッケージ内とサブクラスからアクセス可能
無指定	同パッケージ内からのアクセスのみ可能
private	同クラス内からのアクセスのみ可能



«その他のアクセス制御»

- **フィールド**に使用できるアクセス修飾子は「public」「private」の他に「**protected**」「**無指定**」があります。クラスと同じくアクセス修飾子を何もつけない（無指定）場合、パッケージ外からフィールドへのアクセスはできません。

- protectedを除き、アクセス制御で抑えておくべきポイントは以下のみです。
  - **同じクラス内ではアクセス制限がない**
  - **同パッケージ内の別クラスからのアクセスはprivateでのみ制限**
  - **外部パッケージのクラスからのアクセスは基本的にpublic以外不可**

- **クラス**にもアクセス修飾子「**public**」「**無指定**」が存在します。 アクセス修飾子を何もつけない（無指定）場合、パッケージ外から クラスへのアクセスはできません。 パッケージ外からのアクセスを許す場合にのみpublicをつけます。

クラスに使用可能なアクセス修飾子

アクセス修飾子	説明
public	どこからでもアクセス可能
無指定	同パッケージ内からのアクセスのみ可能

Woodland Academy

Q&A

まとめ

Sum Up

1.Java パッケージの概念と文法：package 文と import 文。

2.カプセル化の概念。

3.Java のカプセル化構文：

① アクセス修飾子：**public**、protected、default、**private**。

② ゲッターとセッター。

Woodland Academy

Woodland Academy

Thank you!

From Seeds to Woodland — Shape Your Future.

Shape Your Future