LightHouseIT

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

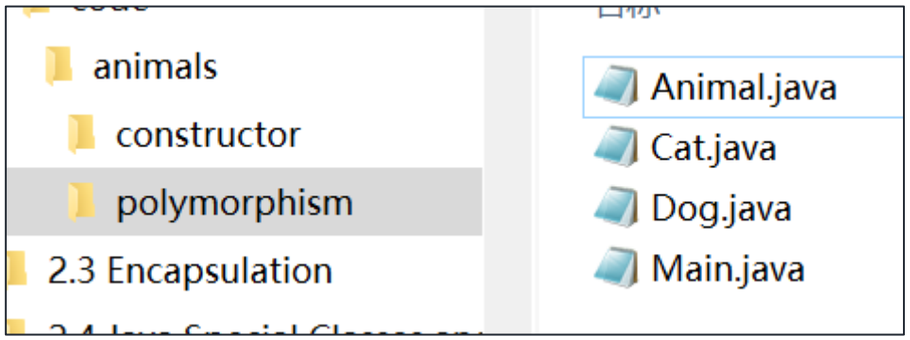
目次

Java パッケージ

- **パッケージ**[Package]とは、Java におけるコードの分類方法の一つです。
- 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。）

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:

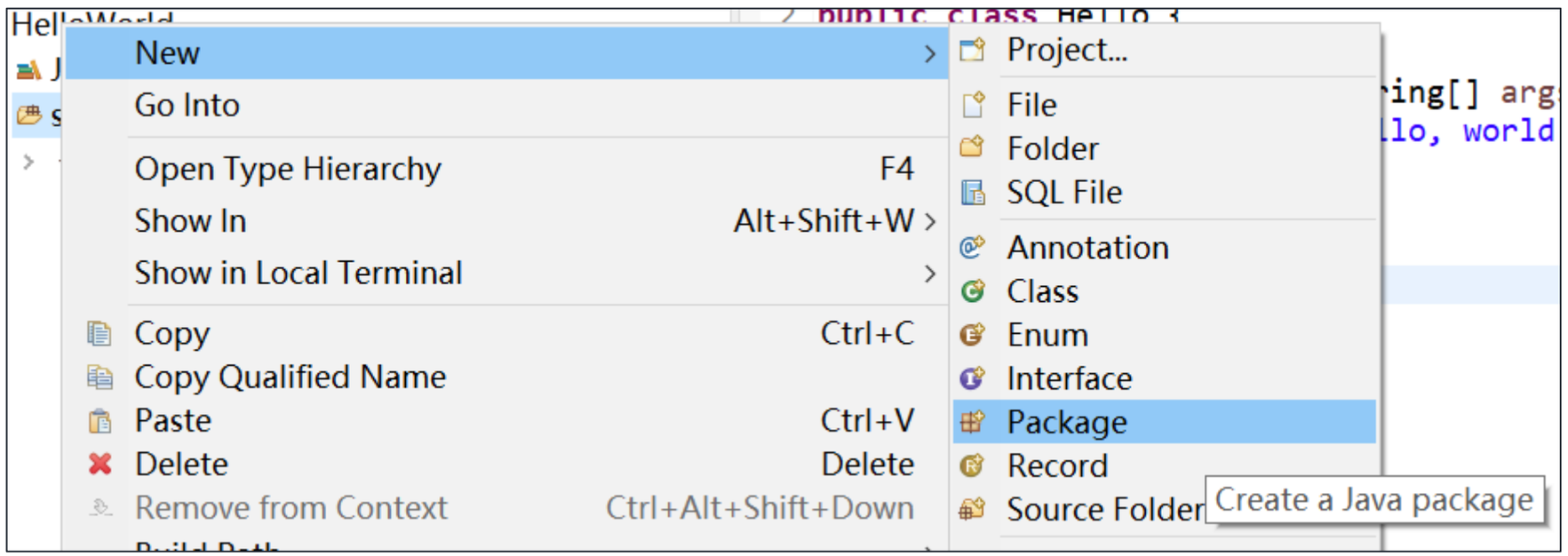
```
package animals.polymorphism;
```

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( );
```

Question and answer

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

- **カプセル化**[encapsulation]とは、オブジェクト指向プログラミングにおいて、オブジェクトの**属性**（プロパティ）と**機能**の一部を**隠蔽**するプロセスです。

- 「隠蔽プロセス」というのは、このオブジェクト（または、このオブジェクトと特定の関係を持つオブジェクト）だけがこれらの属性や機能を使用できるようにします。

- カプセル化の主な機能は以下の 2 つ:
  - オブジェクト自身以外にアクセス**べきない**属性のアクセスを避け、システムの堅牢性を強化します。
  - **必要がない**属性・機能の開示を抑制し、コードの可読性を高めます。

前へ

Example

車のクラスを開発したい。以下の属性・機能のうち、カーオーナーに開示すべきもの、開示すべきでないもの、開示する必要のないものを考えてみてください:

1．車高、車種、色などの属性;
2．車の製造に関わる機能。例：パーツの結合;
3．車の起動・停止に関わる機能;
4．加速度から速度を計算する機能;
5．自動車エンジン改造の機能;
6．価格を変更する機能。

前へ

- このように、公開してはいけない属性や機能があることがわかります。外部クラスがこれらを変更すると他の機能に不具合が生じる可能性があります。例えば、ユーザーが車のエンジンを改造した場合、車の速度を計算する機能が正しく動作できないことがあります。

- 一方、公開する必要のないプロパティや機能もあります。 例えば、自動車を製造する機能や車の速度を計算する機能は、ユーザーにとって何の役にも立ちません。これらの機能をユーザーに知られても、クラスの使い勝手が悪くなるだけ。

- 練習：もし、あなたが**銀行のユーザー**クラスを開発するとしたら、どのような属性や機能を公開する必要があるか? どのようなものは隠蔽すべきか?

目次

LightHouseIT

Java 中のカプセル化

- Java では、**アクセス修飾子**[Access Modifier]によってクラスをカプセル化します。アクセス修飾子は、「どのクラスがメンバー変数やメソッドにアクセスできるか」のような**アクセス権**[Accessibility]を指定できます。

- 最も基本的なアクセス修飾子は、「`public`」と「`private`」の 2 つです。

- メンバ変数やメソッドにアクセス権を追加するには、定義の**前**に指定したい修飾子を記述します:

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

前へ

```
1 public int getCash() {
2     return 0;
3 }
4
5 private String getPassword() {
6     return password;
7 }
```

- public に設定されたメンバ変数やメソッドは他のクラスからアクセスできるに対し、private に設定**されたものはこのクラス内でのみ**アクセスできます:

```
1 public class Main {
2     public static void main(String[] args) {
3         User user = new User();
4         user.username = "Alice";
5         user.password = "123"; // error
6         user.getCash();
7         user.getPassword(); // error
8     }
9 }
```

Try
01011
11010
01011
account
パッケージ

その他のアクセス修飾子

- Java には、public と private の他にもう 2 つのアクセス修飾子があります:

```
1 public int a;
2 protected int b;
3 int c; // default アクセス権
4 private int d;
```

- アクセス修飾子をまとめると、以下のようになる:

修飾子	本クラス	本パッケージ	スブクラス	外部パッケージ
public	○	○	○	○
protected	○	○	○	×
(default)	○	○	×	×
private	○	×	×	×

ゲッターとセッター

- 少し考えてみてください: 商品クラスの「価格」という属性は、顧客に開示されるべきか?

- 同じ属性を**取得**・**設定**するために、異なるアクセス権が必要な場合があります。 しかし、Java では変数のアクセス権が取得と設定の両方に影響します。

- この問題は**ゲッター**[Getter]と**セッター**[Setter]で解決できます。

- アイデア: メンバ変数の取得と設定を 2 つの別々のメソッドとして書いて、アクセス権を**別々**に設定します。

ゲッターとセッターの実現

- まず、この属性のメンバ変数を private に設定し、そのゲッターとセッターをそれぞれ設定する:

```
1 private int price;
2
3 public int getPrice() {
4     return price;
5 }
6
7 private void setPrice(int price_) {
8     price = price_;
9 }
```

- これで、他のクラスは商品オブジェクトの価格を取得することはできますが、設定することはできなくなります。

アクセス権設定のプロセス

- アクセス権設定のプロセスをまとめると以下のようになる:
  1. すべてのメンバ変数とメソッドを「private」に設定する。

  2. すべてのメンバ変数のゲッタとセッタを書く。

  3. すべてのメソッドのアクセス権を要件に応じて調整。

  4. 同じ変数のゲッターやセッターの権限が同じの場合は省略可能。

- カプセル化が正しいと保証するために、最初からアクセス権を「private」に設定することをお勧めします。

Question and answer

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。

LightHouseIT

THANK YOU!