3-15.インターフェース･抽象クラス･ポリモーフィズム

提出する 評定を受ける

■インターフェース･抽象クラス･ポリモーフィズム

インターフェース

これまでの章で、クラスの作成と継承という記述を行ってきました。

クラスというのは、「食事をできる人間」だとか、「走れる車」などのように、プロパティや機能を持った何らかのオブジェクトを作るために作成するものです。

インターフェースというものは、その中で**何らかの機能だけを抽出したような存在**です。

「食事ができる」もしくは「走れる」などの機能だけを持っているというイメージです。

インターフェースを実装する時は、**implements**を使います。

```
class Human implements runnable{
}
```

継承との違いとして、インターフェースは複数のものを実装出来ます。

```
class Human implements runnable, swimmable, flyable{
}
```

一番の特徴は、 **インターフェースの実装対象への依存度が低く、 変更に強いプログラムを構築できる** という点です！

その他、細かな特徴をいくつかあげます。

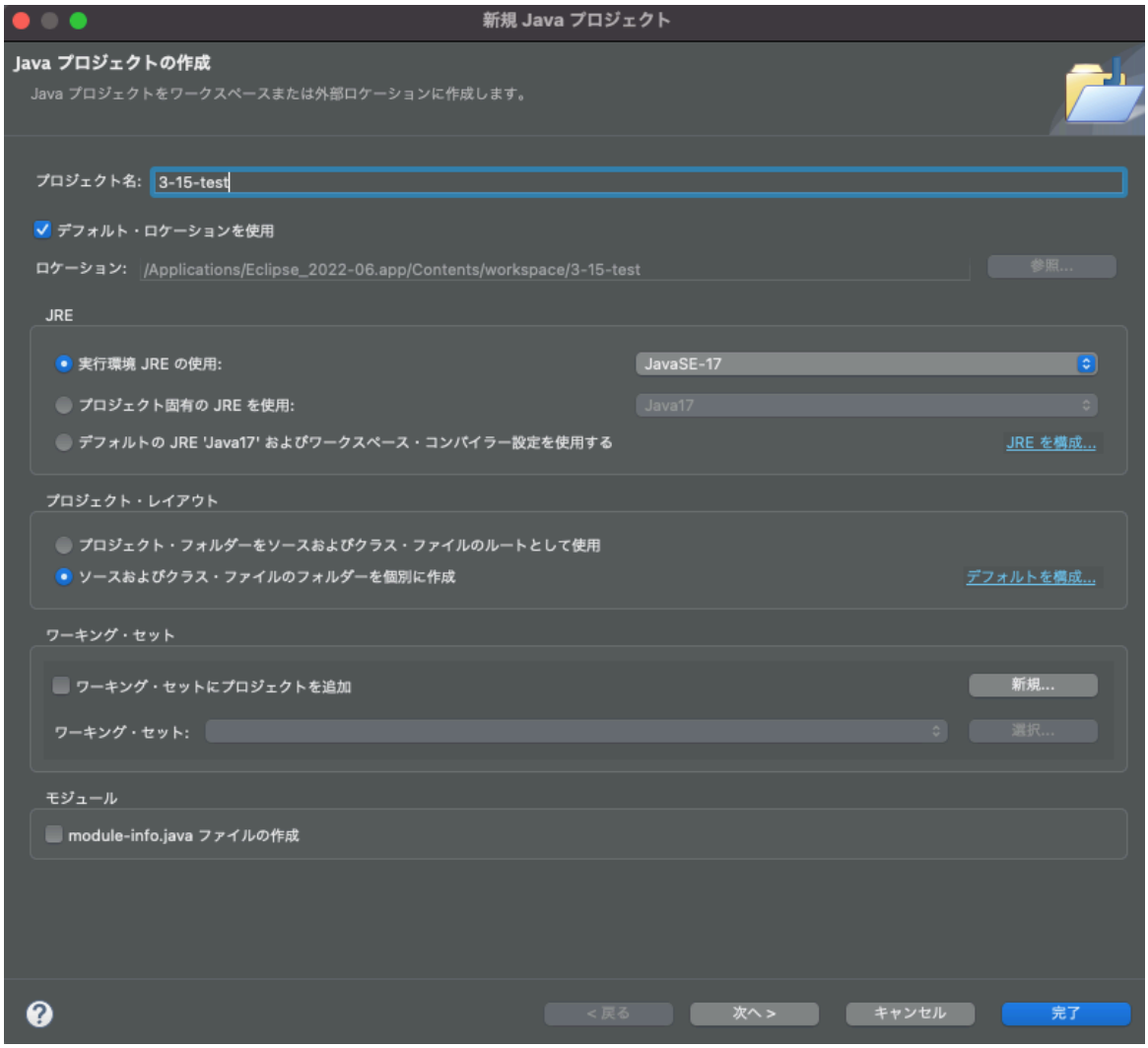
- **処理無しのメソッド** を記述したクラス
- 実装先のクラスは **必ず** インターフェースで宣言されているメソッドをオーバーライドし、
  オーバーライドしたメソッドの実際の処理を記述
- 実装先のクラスはインターフェースに **制御（コントロール）される意味合いを含む**
- 実装先のクラスはインターフェースを **複数** 実装できる

■インターフェースの実装

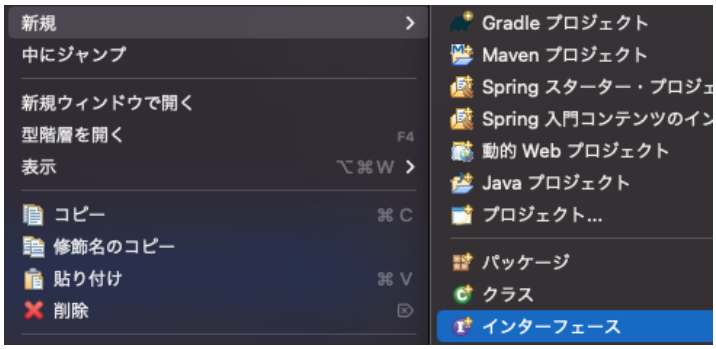
インターフェースを簡単に実装してみます。
最後に課題として提出して頂くので、ご自身で手順通りに実装を行って下さい。

①**「3-15-test」**などの名前で新規Javaプロジェクトを作成して下さい。

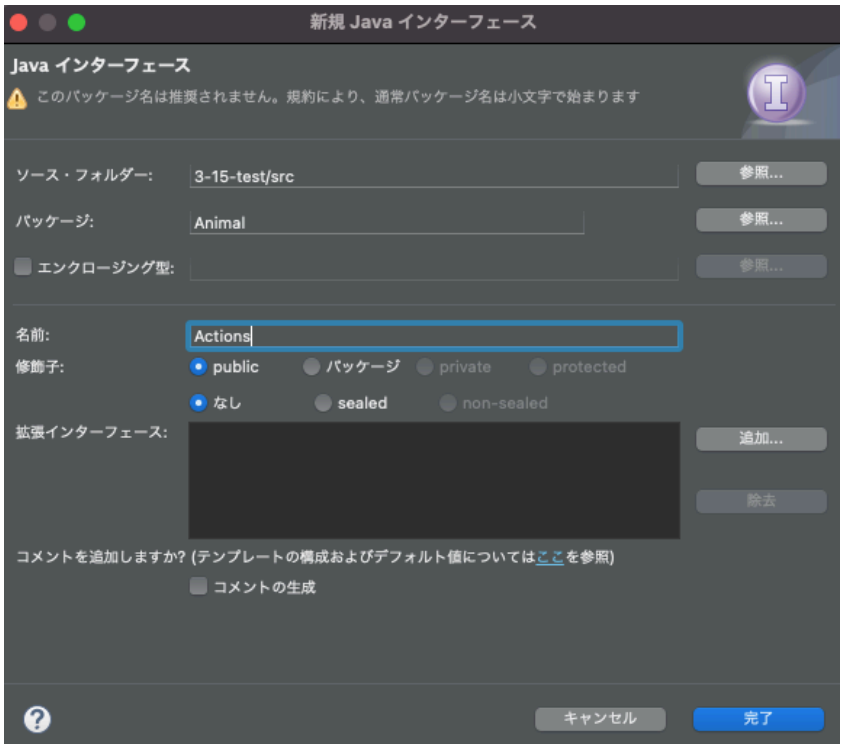
?



②「新規」→「インターフェース」を選択。



③パッケージ名「Animal」、名前を「Actions」にして「完了」を押す



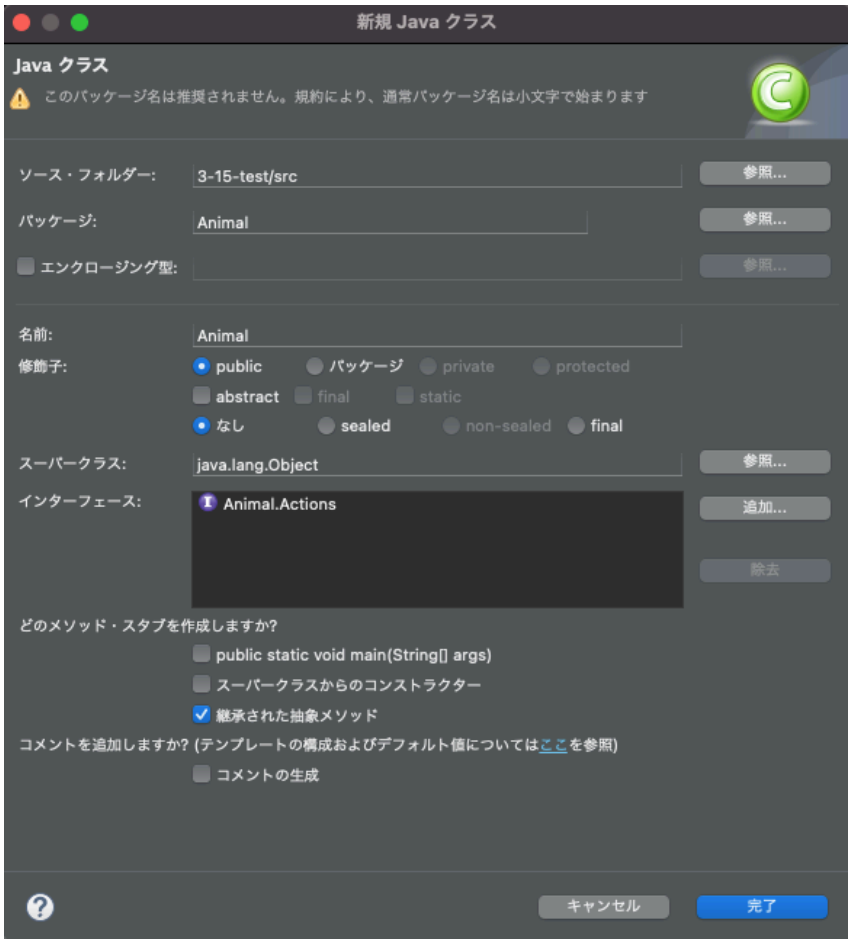
④下記のようなファイルが作成されます。

⑤中身を下記のように記述して下さい。

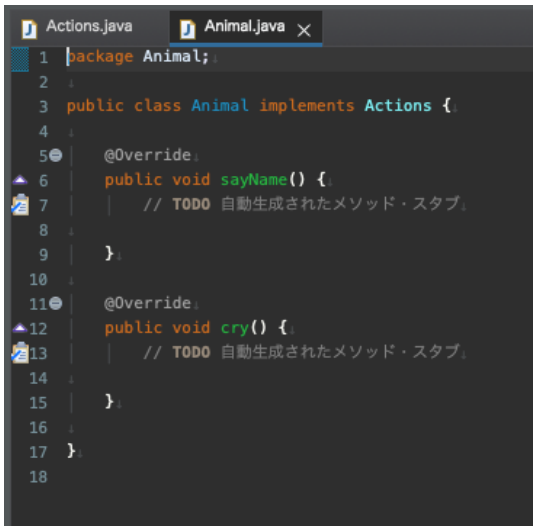
```
package Animal;

public interface Actions {

    /* ----- インターフェースメソッドの {} の処理は何も書かない！ ----- */

 // 名乗りアクション
    public void sayName();


    // 鳴きアクション
    public void cry();

}
```

インターフェースには、上記のように**処理内容の定義をしないメソッド**を記述します。

⑥次に、「Animal」というクラスを作成して下さい。
その時、**設定内の「インターフェース」欄の追加ボタンから、追加させたいインターフェース名(今回はActions)を検索して設定して下さい。**



⑦以下のようなファイルが作成されます。



⑧下記のようにコードを完成させましょう。

```
package Animal;

public class Animal implements Actions {

    private String name;
    private String voice;

    public Animal(String name, String voice) {
        this.name = name;
        this.voice = voice;
    }

        @Override
        public void sayName() {
                System.out.println(name + "です");

        }

        @Override
        public void cry() {
                System.out.println(voice);

        }

}
```

⑨最後に、実行用の「AnimalMain」というクラスを作成し、以下のように記述して実行してみましょう。

```
package Animal;

public class AnimalMain {

        public static void main(String[] args) {
                Animal animal = new Animal("猫","にゃあ");
                animal.sayName();
                animal.cry();
        }

}
```

実行結果

```
猫です
にゃあ
```

インターフェイスをうまく活用することで、大規模開発の際の実装漏れを防げます。

インターフェイスは、そのインターフェイス内で定義したメソッドの実装を保証します。**インターフェイスで定義された抽象メソッドが未実装の場合、コンパイルエラーが発生するのです。**

よって、あらかじめ必要なメソッドをインターフェイスに定義しておけば、実装漏れによるバグの発生を防ぐことが出来ます。

抽象クラス

抽象クラスは、インターフェースと通常のクラスの中間 のようなクラスです。

インターフェースではメソッドのみを定義しましたが、抽象クラスでは通常のクラスと同様に、 フィールド や コンストラクタ 、 メソッド等まで範囲を広げて定義できるもの、という点を抑えて下さい。

特徴としては以下のような点があります。

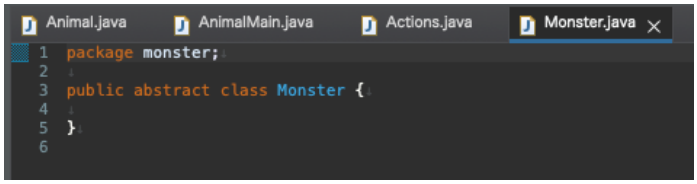
- 「継承」と同様に、複数の抽象クラスは継承不可
- **抽象クラス内の `abstract` が付いたメソッド（抽象メソッド）**は、
  この抽象クラスを継承するクラスでのオーバーライドと中身の処理の実装が必須
- **必ず** サブクラスのインスタンスを生成して使用する(抽象クラスを直接インスタンス化はしない)

■抽象クラスの継承

まずは簡単に実装してみましょう。

①先程のプロジェクトフォルダ内で新規クラスの作成を行って下さい。
パッケージ名は「monster」、クラス名は「Monster」にして下さい。
その際、「修飾子」のオプションで「abstract」にチェックを入れます。

②以下のようなファイルが作成されます。



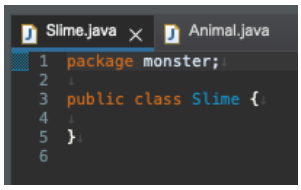
③下記のようにコードを記述して下さい。

```
package monster;

public abstract class Monster {

    private String name;
    private String skill;

    public Monster(String name, String skill) {
        this.name = name;
        this.skill = skill;
    }

 public abstract void sayName();

 public abstract void attack();

    protected String getName() {
```

```
        return name;
    }

    protected String getSkill() {
        return skill;
    }
}
```

インターフェースと違い、抽象クラスではプロパティ、コンストラクタも定義しています。
また、クラス名やメソッド名には修飾子「abstract」を付与しておきます。

④Monsterクラスを継承した、Slimeクラスを作成します。
同じmonsterパッケージ内でクラスの新規作成を行い、「Slime」クラスとして下さい。



⑤以下のようにコードを記述して下さい

```
package monster;

public class Slime extends Monster{

        public Slime(String name, String skill) {
                super(name, skill);
        }

        @Override
        public void sayName() {
                System.out.println(super.getName() + "があらわれた！");
        }

        @Override
        public void attack() {
                System.out.println(super.getSkill() + "で攻撃！");
        }
}
```

抽象クラスの継承は、普通の継承と同じように**「extends」**で行います。
また、インターフェースと同様に、抽象クラスで定義して抽象メソッドは必ずオーバーライドを行います。

また、継承した時は親クラスのコンストラクタを実行するのは必須なので、**super(name, skill);** によってそれを行っています。

最後に、実行用のクラス**「MonsterMain」**クラスを作成し、以下のように記述、実行しましょう。

```
package monster;

public class MonsterMain {

        public static void main(String[] args) {
                Monster monster = new Slime("スライム","たいあたり");
                monster.sayName();
                monster.attack();
        }
}
```

実行結果

```
スライムがあらわれた！
たいあたりで攻撃！
```

以上のように、抽象クラスもインターフェースと同じように、プロパティ･コンストラクタ、メソッドを同じような型で使うことを統一させる役割で使用されます！

ポリモーフィズム

ここまでの抽象クラスやインターフェースなどを用いて、同じ型で様々な機能を実装させる手法を**ポリモーフィズム(多態性)**と呼びます。

例えば、先程のmonsterパッケージ内で「Dragon」という新たなクラスを作成してみましょう。

```
package monster;

public class Dragon extends Monster{
 public Dragon(String name, String skill) {
 super(name, skill);
 }

 @Override
 public void sayName() {
 System.out.println("ボスの" + super.getName() + "があらわれた！");
 }
 @Override
 public void attack() {
 System.out.println(super.getSkill() + "で全体を攻撃！");
 }

}
```

こちらを、MonsterMainクラスで続けて実行してみます。

```
package monster;

public class MonsterMain {

	public static void main(String[] args) {
		Monster monster = new Slime("スライム","たいあたり");
		monster.sayName();
		monster.attack();

		monster = new Dragon("ドラゴン","火の息");
		monster.sayName();
		monster.attack();
	}
}
```

実行結果

```
スライムがあらわれた！
たいあたりで攻撃！
ボスのドラゴンがあらわれた！
火の息で全体を攻撃！
```

大きな特徴としては、メソッドの実行(sayName、attack)の命令は統一出来ている点です！
全体の枠と命令を統一した上で、それぞれで別の動きをもたせることが出来るのがポリモーフィズムのメリットとなります。

課題

今回使用したプロジェクト内(例:3_15-test)で下記の追加記述を行って、プロジェクト全体を提出して下さい

①Animalパッケージ内で、追加のインターフェースメソッドを1つ実装して実行させる
②Monsterパッケージ内で、追加のモンスタークラスを1つ実装して実行させる

その他試したいコードを追加や修正頂いても構いません。

評定概要

可視グループ: すべての参加者

学生から秘匿	No
参加者	75
提出	51
要評定	4