23．Javaの多様性

　23.1メソッドと多様性

Javaのメソッドは、オブジェクト指向プログラミングの基本的な概念の一つであり、オブジェクトが行う操作を定義するために使用されます。Javaのメソッドには多様性があります。以下に説明します。

1. メソッドのオーバーロード：Javaでは、同じ名前のメソッドを複数定義することができます。この機能は、オーバーロードと呼ばれます。オーバーロードされたメソッドは、引数の数、型、順序が異なる場合に使用されます。
2. メソッドのオーバーライド：Javaでは、親クラスのメソッドを子クラスで再定義することができます。この機能は、オーバーライドと呼ばれます。オーバーライドされたメソッドは、親クラスのメソッドを置き換えて、子クラスのメソッドが実行されます。
3. 関数型インタフェース：Java 8から、関数型インタフェースが導入されました。関数型インタフェースは、1つの抽象メソッドを持つインタフェースであり、ラムダ式やメソッド参照を使用して、関数のように扱うことができます。
4. デフォルトメソッド：Java 8から、インタフェースにデフォルトメソッドが追加されました。デフォルトメソッドは、インタフェースに実装されるメソッドであり、既存の実装を変更せずに新しい機能を追加することができます。
5. スタティックメソッド：Java 8から、インタフェースにスタティックメソッドが追加されました。スタティックメソッドは、インスタンスを作成せずに呼び出すことができ、インタフェースの共通の機能を提供することができます。

これらの多様なメソッドの種類により、Javaのメソッドは非常に柔軟で、開発者が異なるニーズに応じてメソッドを定義することができます。

23．1.1例）メソッドのオーバーロードのサンプル

public class Calculator {

public int add(int a, int b) {

return a + b;

}

public int add(int a, int b, int c) {

return a + b + c;

}

}

// 使用例

Calculator calc = new Calculator();

int result1 = calc.add(3, 4);

int result2 = calc.add(3, 4, 5);

この例では、Calculatorクラスにはaddメソッドが2つ定義されています。1つは2つの引数を受け取り、もう1つは3つの引数を受け取ります。このように、引数の数が異なる同じ名前のメソッドを定義することができます。

23.2　クラスと多様性

Javaでは、クラスの継承を使用して、既存のクラスを拡張することができます。子クラスは親クラスのメソッドやフィールドを利用できるため、コードの再利用性が高くなります。

例）public class Animal {

public void makeSound() {

System.out.println("The animal makes a sound");

}

}

public class Cat extends Animal {

@Override

public void makeSound() {

System.out.println("Meow!");

}

}

// 使用例

Animal animal = new Animal();

animal.makeSound(); // "The animal makes a sound"が出力される

Cat cat = new Cat();

cat.makeSound(); // "Meow!"が出力される

23.3インターフェースの実装

public interface Flyable {

void fly();

}

public interface Swimmable {

void swim();

}

public class Bird implements Flyable {

@Override

public void fly() {

System.out.println("The bird is flying");

}

}

public class Fish implements Swimmable {

@Override

public void swim() {

System.out.println("The fish is swimming");

}

}

// 使用例

Bird bird = new Bird();

bird.fly(); // "The bird is flying"が出力される

Fish fish = new Fish();

fish.swim(); // "The fish is swimming"が出力される

23.4抽象クラス

public abstract class Shape {

public abstract double area();

}

public class Rectangle extends Shape {

private double length;

private double width;

public Rectangle(double length, double width) {

this.length = length;

this.width = width;

}

@Override

public double area() {

return length \* width;

}

}

// 使用例

Shape shape = new Rectangle(3, 4);

double area = shape.area(); // 12.0が返される

23.5内部クラスは、

別のクラスの中に定義されたクラスであり、外部のクラスのメンバー変数やメソッドにアクセスできることが特徴です。内部クラスは多様性を持つことができ、以下のような種類があります。

23．5.1ローカルクラス

ローカルクラスは、メソッド内で定義されたクラスであり、そのメソッド内でしか使用できません。ローカルクラスは、メソッド内の変数にアクセスできるため、メソッド内で使用されるオブジェクトをより柔軟に扱うことができます。

public class Outer {

void method() {

class Local {

void print() {

System.out.println("Hello from local class");

}

}

Local l = new Local();

l.print();

}

}

// 使用例

Outer o = new Outer();

o.method(); // "Hello from local class"が出力される

この例では、Outerクラスのmethodメソッド内でLocalクラスを定義しています。Localクラスは、methodメソッド内でしか使用できないため、Outerクラスの外部からはアクセスできません。ローカルクラスは、外部の変数をキャプチャすることができるため、上記の例ではLocalクラスがmethodメソッド内で使用される変数にアクセスすることができます。