21.１ クラスと変数へのアクセス権

Javaにおけるアクセス権は、クラス、変数、メソッドなどのメンバーに対して、どの範囲で他のクラスからアクセスできるかを制御する機能です。Javaには以下の4つのアクセス権があります。

1. public: 他のクラスから自由にアクセスできます。
2. private: 同じクラス内からのみアクセスできます。
3. protected: 同じパッケージ内、またはサブクラスからのみアクセスできます。
4. デフォルト(指定なし): 同じパッケージ内からのみアクセスできます。

これらのアクセス権を使用することで、プログラマーはプログラムのセキュリティと保守性を向上させることができます。たとえば、変数やメソッドをpublicにすることで、外部からのアクセスが容易になりますが、それによって不正なアクセスや誤った使用が発生する可能性があります。一方、privateにすることで、外部からのアクセスを制限することができますが、クラス内部でしか使用できなくなるため、そのクラスのメンバーを操作する必要がある場合は、publicなメソッドを提供することで解決することができます。

以下にJavaのアクセス権を使用したサンプルコードを示します。

// パッケージ名 com.example

package com.example;

// クラス名 SampleClass

public class SampleClass {

// privateな変数

private int privateVar;

// publicなメソッド

public void setPrivateVar(int value) {

this.privateVar = value;

}

// publicなメソッド

public int getPrivateVar() {

return this.privateVar;

}

// protectedなメソッド

protected void protectedMethod() {

System.out.println("This is a protected method");

}

// デフォルトのアクセス権限

void defaultMethod() {

System.out.println("This is a default method");

}

}

上記の例では、SampleClassというクラスがあります。クラス内には、privateな変数とpublicなメソッドがあります。また、protectedなメソッドとデフォルトのアクセス権限を持つメソッドも定義されています。

この場合、SampleClassのprivateVar変数に直接アクセスすることはできません。代わりに、publicなsetPrivateVarメソッドを使用して値を設定する必要があります。同様に、privateVar変数の値を取得するには、publicなgetPrivateVarメソッドを使用します。

protectedMethodメソッドは、同じパッケージ内のクラスや、SampleClassを継承したサブクラスからのみアクセスできます。defaultMethodは、同じパッケージ内のクラスからのみアクセスできます。

以下にJavaのクラス間でのアクセス権を使用したサンプルコードを示します。

// パッケージ名 com.example

package com.example;

// SampleClassクラス

public class SampleClass {

// privateな変数

private int privateVar;

// publicな変数

public int publicVar;

// publicなメソッド

public void setPrivateVar(int value) {

this.privateVar = value;

}

// publicなメソッド

public int getPrivateVar() {

return this.privateVar;

}

// protectedなメソッド

protected void protectedMethod() {

System.out.println("This is a protected method");

}

// デフォルトのアクセス権限

void defaultMethod() {

System.out.println("This is a default method");

}

}

// パッケージ名 com.example

package com.example;

// 別のクラス

public class AnotherClass {

public static void main(String[] args) {

// SampleClassのインスタンスを生成

SampleClass sample = new SampleClass();

// publicな変数には直接アクセスできる

sample.publicVar = 100;

// privateな変数には直接アクセスできないため、publicなメソッドを使用する

sample.setPrivateVar(200);

// privateな変数の値を取得するためにpublicなメソッドを使用する

System.out.println(sample.getPrivateVar());

// protectedなメソッドにはアクセスできない

// sample.protectedMethod();

// デフォルトのアクセス権限のメソッドには同じパッケージ内からはアクセスできる

sample.defaultMethod();

}

}

上記の例では、SampleClassというクラスと、AnotherClassという別のクラスがあります。AnotherClassからSampleClassのインスタンスを生成し、publicな変数には直接アクセスできます。一方、privateな変数には直接アクセスできませんが、publicなsetPrivateVarメソッドを使用して値を設定し、publicなgetPrivateVarメソッドを使用して値を取得することができます。

また、SampleClassのprotectedMethodメソッドにはアクセスできませんが、defaultMethodメソッドには同じパッケージ内からアクセスすることができます。

## 21.2　abstract

abstract class Shape {

private String color;

public Shape(String color) {

this.color = color;

}

public abstract double getArea();

public String getColor() {

return color;

}

}

class Circle extends Shape {

private double radius;

public Circle(String color, double radius) {

super(color);

this.radius = radius;

}

public double getArea() {

return Math.PI \* Math.pow(radius, 2);

}

}

class Rectangle extends Shape {

private double width;

private double height;

public Rectangle(String color, double width, double height) {

super(color);

this.width = width;

this.height = height;

}

public double getArea() {

return width \* height;

}

}

public class Main {

public static void main(String[] args) {

Shape shape1 = new Circle("Red", 2.0);

Shape shape2 = new Rectangle("Blue", 2.0, 4.0);

System.out.println("Area of " + shape1.getColor() + " circle: " + shape1.getArea());

System.out.println("Area of " + shape2.getColor() + " rectangle: " + shape2.getArea());

}

}

この例では、**Shape**という抽象クラスが定義されています。このクラスは、色のフィールドを持ち、**getArea()**メソッドを抽象メソッドとして宣言しています。**Circle**と**Rectangle**クラスは、それぞれ**Shape**を継承しています。これらのクラスでは、**getArea()**メソッドをオーバーライドして、円と長方形の面積を計算するためのロジックを提供しています。

**Main**クラスでは、**Circle**と**Rectangle**オブジェクトを作成して、それらの面積を計算しています。**Shape**型の変数を使用して、**Circle**と**Rectangle**オブジェクトの両方を参照することができます

// 抽象クラスの定義

abstract class Animal {

// 抽象メソッドの定義

public abstract void makeSound();

// 通常のメソッドの定義

public void eat() {

System.out.println("This animal is eating.");

}

}

// Animalクラスを継承するDogクラスの定義

class Dog extends Animal {

// 抽象メソッドを実装

public void makeSound() {

System.out.println("Bark bark!");

}

}

24.3

// Animalクラスを継承するCatクラスの定義

class Cat extends Animal {

// 抽象メソッドを実装

public void makeSound() {

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

}

}

// メインメソッド

public class Main {

public static void main(String[] args) {

Animal myDog = new Dog();

Animal myCat = new Cat();

myDog.makeSound(); // 出力：Bark bark!

myDog.eat(); // 出力：This animal is eating.

myCat.makeSound(); // 出力：Meow meow!

myCat.eat(); // 出力：This animal is eating.

}

}

24.3 final

public class MyClass { final int MAX\_VALUE = 100; // final変数 public static void main(String[] args) { final double PI = 3.14159; // final変数 System.out.println("The maximum value is: " + MAX\_VALUE); System.out.println("The value of PI is: " + PI); } }

上記の例では、**MAX\_VALUE** と **PI** は両方とも **final** 変数として宣言されています。 **MAX\_VALUE** は、 **MyClass** クラスのインスタンス変数として定義され、 **PI** は **main** メソッド内でローカル変数として定義されていますpublic class MyClass {

final int MAX\_VALUE = 100; // final変数

public static void main(String[] args) {

final double PI = 3.14159; // final変数

System.out.println("The maximum value is: " + MAX\_VALUE);

System.out.println("The value of PI is: " + PI);

}

}

上記の例では、**MAX\_VALUE** と **PI** は両方とも **final** 変数として宣言されています。 **MAX\_VALUE** は、 **MyClass** クラスのインスタンス変数として定義され、 **PI** は **main** メソッド内でローカル変数として定義されています。

**final** 変数は、宣言時に初期化されなければなりません。また、値の変更ができないことを意味します。上記の例では、 **MAX\_VALUE** と **PI** はいずれも初期化され、値を変更することができない定数として使用されます。

フォームの始まり