24.OBJECTクラス

Objectクラスは、Javaのすべてのクラスのスーパークラスであり、

例として、class ObjectString｛｝は　class ObjectString extends Object｛｝と同じです

我々が定義したクラスはObjectクラスの継続であり、Objectクラスのメンバーの

toString、equals　methodを再定義（overriding）を行えます。

すべてのJavaオブジェクトはObjectクラスの直接のサブクラスであるため、Objectクラスで定義されたメソッドを利用することができます。

Objectクラスで定義されているメソッドには、以下のようなものがあります。

1. equalsメソッド オブジェクトの等価性を比較するメソッドであり、オブジェクト同士を比較して等しい場合はtrueを返し、そうでない場合はfalseを返します。equalsメソッドは、オブジェクト同士を比較するためにオーバーライドされることが多く、オブジェクト同士を比較するための基準を定義することができます。
2. hashCodeメソッド オブジェクトのハッシュコードを返すメソッドであり、同じオブジェクトに対しては常に同じハッシュコードを返します。hashCodeメソッドは、オブジェクトをハッシュテーブルに格納する際に使用されることが多く、equalsメソッドと一緒にオーバーライドされることが多いです。
3. toStringメソッド オブジェクトの文字列表現を返すメソッドであり、デフォルトでは、クラス名とオブジェクトのハッシュコードを返します。toStringメソッドは、オブジェクトのデバッグやログ出力などで利用されることが多く、オブジェクトの状態を文字列として出力するためにオーバーライドされることがあります。
4. cloneメソッド オブジェクトを複製するメソッドであり、Objectクラスでは、クローンをサポートしないため、このメソッドはクラスでオーバーライドされる必要があります。
5. finalizeメソッド オブジェクトがガベージコレクションされる際に呼び出されるメソッドであり、オブジェクトが不要になったときに何らかの後処理を行うためにオーバーライドされることがあります。

Objectクラスは、Javaのすべてのクラスに共通する基本的な機能を提供するため、Javaのオブジェクト指向プログラミングの基礎となる重要なクラスです

OBJECT API 文書

<http://docs.oracle.com/javase/7/docs/api/java/lang/Object.html>

１．toStringのoverrridding

package javaProjet;

// objectクラスは、Javaのすべてのクラスのスーパークラスであり

//すべてのJavaオブジェクトはObjectクラスの直接のサブクラスであるため、

//Objectクラスで定義されたメソッドを利用することができます。

//OBJECT API 文書

//http://docs.oracle.com/javase/7/docs/api/java/lang/Object.html

class ObjectString{

private String name;

private int age;

public ObjectString(String name, int age) {

this.name = name;

this.age = age;

System.out.println("this.name = " + this.name);

System.out.println("this.age = " + this.age);

}

@Override

public String toString() {

return "Person [name=" + name + ", age=" + age + "]";

}

}

public class ObjectToString {

public static void main(String[] args) {

ObjectString ObjtString = new ObjectString("John Doe", 30);

System.out.println(ObjtString.toString());

}

}

2．equalsのoverrridding

package javaProjet;

class ObjectEqualName{

String name;

ObjectEqualName(String name){

this.name = name;

}

public boolean equals(Object obj) {

ObjectEqualName \_obj = (ObjectEqualName)obj;

return name == \_obj.name;

}

}

public class ObjectEqual {

public static void main(String[] args) {

ObjectEqualName s1 = new ObjectEqualName("野村");

ObjectEqualName s2 = new ObjectEqualName("野村");

// String s1 = "nomura";

// String s2 = "nomura";

System.out.println(s1 == s2); //Stringの比較は保証できない

System.out.println(s1.equals(s2));

String s5 = "Hello";

String s6 = new String("Hello");

System.out.println(s5.equals(s6)); // true

}

}

ただし、Javaでは、**toString()** メソッドはオブジェクトの振る舞いを変えないため、必ずしもオブジェクトの状態を表現するために適した文字列を返すわけではありません。**toString()** メソッドをオーバーライドすることで、より有用な出力を生成できますが、必要に応じて独自のメソッドを定義して、オブジェクトを表現するためのカスタム文字列を生成することもできまの

Javaの**equals()**メソッドは、2つのオブジェクトが同値であるかどうかを比較するために使用されます。以下では、**equals()**メソッドについて詳しく説明します。

**equals()**メソッドのシグネチャは以下のようになっています：

javaCopy code

public boolean equals(Object obj)

このメソッドは、引数として与えられたオブジェクト（**obj**）と、メソッドを呼び出したオブジェクト自体とを比較します。通常、オーバーライドされることが多く、オブジェクトのクラスに固有の比較方法を実装します。

**equals()**メソッドのデフォルトの実装では、オブジェクトの参照（メモリ上の場所）を比較します。つまり、2つのオブジェクトが同じメモリ位置に格納されている場合にのみ、**equals()**メソッドは**true**を返します。これは、**equals()**メソッドをオーバーライドしない限り、Javaの全てのクラスに共通です。

ただし、多くの場合、プログラムでは異なるオブジェクトでも同じ値を持つかどうかを比較する必要があります。そのため、多くのJavaクラス（例えば、**String**クラス）では、**equals()**メソッドがオーバーライドされています。

オブジェクトが同値であるかどうかを判断するために**equals()**メソッドをカスタマイズする場合、一般的な実装方法は以下の通りです：

1. 引数のオブジェクトが**null**でないかを確認します。**null**である場合、**equals()**メソッドは**false**を返すか、例外をスローします。
2. 引数のオブジェクトが**this**と同じインスタンスであるかを確認します。もし同じインスタンスである場合、**equals()**メソッドは**true**を返します。
3. 引数のオブジェクトが比較可能な型であるかを確認します。もし比較できない型である場合、**equals()**メソッドは**false**を返します。
4. 引数のオブジェクトと**this**のフィールドを比較して、等価性を判断します。フィールドが等しい場合、**equals()**メソッドは**true**を返します。

具体的な実装はクラスによって異なります。たとえば、**String**クラスでは、文字列の内容を比較します。**equals()**メソッドをオーバーライドする場合、**hashCode()**メソッドも一緒にオーバーライドする必要があります。これは、**equals()**メソッドが**true**を返すオブジェクトは同じハッシュコードを持つべきであるためです。