Share

08 OOP - その 2

Created by GT F
Last updated 2019-08-13

継承

クラスの記述するときは基底クラス（base class）を指定できます。派生クラスは基底クラスから導出されます。クラスは**単一継承**のみに対応します。

⚠ 派生クラスは public / protected 属性のみアクセス可能。

派生した（引き続ぐ方）クラスは「サブクラス」と呼びます。引き続がれた方は「スーパークラス」あるいは「親クラス」と呼びます。キーワード： `extends` を使用して、クラスを継承します。

```
// Parent.java 親クラス
public class Parent {
  /**姓*/
  public String familyName;
  /**名*/
  private String givenName;
  /**財産*/
  protected int money;
  /**メソッド*/
  public String hello() {
      System.out.println("say hello");
      return "123";
  }
```

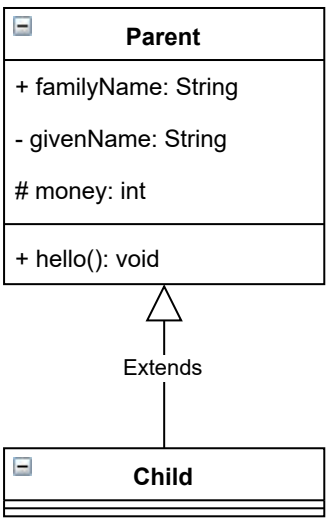
```
}
// Child.java サブクラス
public class Child extends Parent {

}
// Kicker.java
public class Kicker {
    public static void main(String...args) {
        Child child = new Child();
        System.out.println(child.familyName);
        // String value = child.hello();
        // System.out.println(value);
        System.out.println(child.hello());
    }
}
```

質問：Parentクラスの３つフィールドを保有しています。Childクラス２つフィールドを継承します。親クラスの`givenName`は`private`の為、継承しません。

UML

UML.Class図に、継承の記載方法



IS-A 関係

クラスBがクラスAを継承する場合、日本語で「BはAの一種である」を成り立つ。クラスCがクラスBを継承するのであれば、「C is B」だけではなく、「C is A」も成り立ちます。このような文で表せる関係は「IS-A」関係と呼び出す。

```
// Shape.java 図形
public class Shape {}
// Square.java 正方形
public class Square extends Shape {}
```

```java
// Kicker.java
public class Kicker {
    public static void main(String...args) {
        Shape p = new Shape();
        Shape c = new Square(); // 正方形は図形の一種である
    }
}
```

override

オーバーライトとは、サブクラス内で、スーパークラスで定義しているメソッドを同じ名称に再定義することです。サブクラスがインスタンス化され、オーバーライドされたメソッドが呼ばれた際には、サブクラスに再定義されたメソッドを優先的に呼び出される。

```java
// Parent.java
public class Parent {
    public void hello() {
        System.out.println("parent method");
    }
}
// Child.java
public class Child extends Parent {
    @Override // 親クラスの強制的にOverrideする意味（アノテーション）
    public void hello() {
        System.out.println("child method");
    }
}
// Kicker.java
public class Kicker {
    public static void print(Parent v) {
        v.hello(); // Override実際発生する箇所
    }
    public static void main(String...args) {
        print(new Parent()); // parent method
        print(new Child()); // child method ⇒ 派生クラスは直接に渡す可能
    }
}
```

super & this

`this`は、自分の**インスタンス**を指すキーワードです。`super`は親クラスのクラスメンバを指しています。

```java
// Parent.java
public class Parent {
    protected int b; // 注意
    private int a;
    public void hello() {
        System.out.println("parent method");
    }
```

```
}
// Child.java
public class Child extends Parent {
    protected int b; // 注意
    public Child() {
        super(); // 親クラスのコンストラクタを呼び出す
    }
    @Override
    public void hello() {
        super.b = 100;
        this.b = 200;
        System.out.println("invoked from child");
    }
}
```

親クラスのメソッドを呼び出す

```
// Parent.java
public class Parent {
    public void hello() {
        System.out.println("parent method");
    }
}
// Child.java
public class Child extends Parent {
    @Override
    public void hello() {
        super.hello(); // 親クラスのhello()メソッドを最呼び出す
        System.out.println("child method");
    }
}
// Kicker.java
public class Kicker {
    public static void main(String...args) {
        Child c = new Child();// cのタイプはChild又はParentであること
        c.hello(); // child method ⇒ 派生クラスは直接に渡す可能
    }
}
```

型判断

型の判別は演算子`instanceof`を使用して判別します。

演算子	説明
instance of	あるオブジェクトは指定するクラス型であるかを判断する

実際`instanceof`を使用例です。

```
public static void main(String[] args) {
    Parent p = new Parent();
```

```
    Child c = new Child();
    System.out.print(p instanceof Child); // false
    System.out.print(c instanceof Parent); // true
}
```

Object

全てのクラスはObjectクラスを継承しています。このクラスはJava参照型（クラス型）を基本機能を定義しています。よく以下２つメソッドをオーバーライドします。

1. toString()：オブジェクトの文字列の表現を返す。
2. equals(Object another)：引数のオブジェクトは等し（ひとしい）かを返す。

toString

インスタンスを文字列に変換して返します。toString()をオーバーライドしない場合、インスタンスのHashCode（メモリアドレス）を返却します。この場合人間は読めません。以下例を実施してください。

```
public class Kicker {
    public static class InnerClassA {
        public String name;
        public InnerClassA(String name) {
            this.name = name;
        }
    }
    public static class InnerClassB {
        public String name;
        public InnerClassB(String name) {
            this.name = name;
        }
        @Override // Objectクラスのto String()をオーバーライドする
        public String toString() {
            return "クラス名称=" + name;
        }
    }

    public static void main(String...args) {
        // ?? Object.toString()を呼び出す
        System.out.println(new InnerClassA("innerClassA"));
        // ?? InnerClassB.toString()を呼び出す
        System.out.println(new InnerClassB("innerClassB"));
    }
}
```

equals

演算子`==`はオブジェクトを比較し、同じメモリアドレス情報が一致するかを判断します。

```
String str1 = new String("string 1");
String str2 = new String("string 1");
System.out.println(str1 == str2); // false:メモリアドレスは一致しません。
```

`equals`メソッドは各クラスでオーバーライドすることができます。Stringクラスは`equals`をオーバーライドして、メモリアドレス比較ずに、文字列の内容を比較します。

```
String str1 = new String("string 1");
String str2 = new String("string 1");
System.out.println(str1.equals(str2)); // true:文字列内容は一致します。
```

継承＆型変換（キャスト）

親クラス⇒子クラスキャストできる前提条件は該当オブジェクトの実型はキャスト対象クラス

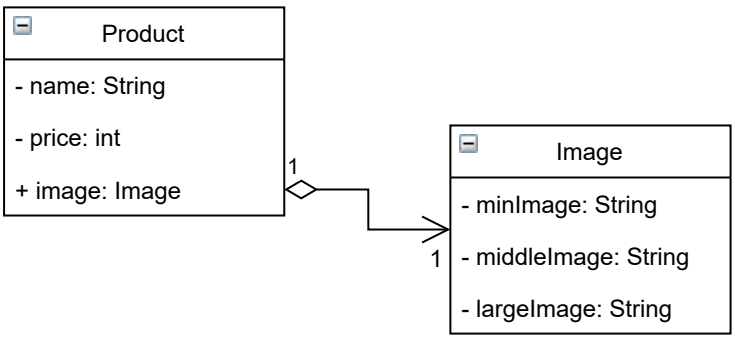
```
public static void main(String...args) {
    Object str = new String("DCNET");
    String value1 = (String)str;
    String value2 = String.class.cast(value1);
    String excep = (int)str; // エラー：変数strの実型はStringである。
}
```

集約関係

あるオブジェクトが他のオブジェクトの一部であったり、他のオブジェクトをもっていたりする関係を**集約**といいます。アマゾンの各商品にて写真が保有しています。各写真は大中小３サイズの画像を保存しています。以下通りクラスを定義する。

```
public class Image {
    private String minImage;
    private String middleImage;
    private String largeImage;
    // getter setter
    // ....
}
public class Product {
    /**名称*/
    private String name;
    /**価格*/
    private int price;
    /**商品画像*/
    private Image image;
    // getter setter
    // ...
}
```

UMLの表現：ProductがImageを「**所有する**」。



継承不可

`final`キーワードが指定された場合、該当クラスは継承不可になります。

```
public final class NoChild {} // 該当クラスを継承不可
```

質問：以下コードのコンパイルエラー理由を教えてください。

```
// Parent.java
public final class Parent {}
// Child.java
public class Child extends Parent {}
```

質問

質問１：以下コードの出力するを回答してください。

```
String x = "a";
String y = "a";
System.out.println(x == y); // true ? false? 原因は ?
System.out.println(x.equals(y)); // true ? false? 原因は ?

String empty = "";
String nullString = null;
System.out.println(nullString.equals(empty)); // true ? false? 原因は ?
```

質問２：以下コードの出力を回答してください。

```
// Employee.java
public class Employee {
    private String id;
    public Employee(String id) {
        this.id = id;
    }
    @Override
    public boolean equals(Object another) {
        if (another instanceof Employee) {
```

```
            Employee an = (Employee) another;
            return an.id.equals(id);
        }
        return false;
    }
}
// Kicker.java
public class Kicker {
    public static void main(String...args) {
        Employee x = new Employee("1234");
        Employee y = new Employee("1234");
        System.out.println(x == y); // true ? false? 原因は ?
        System.out.println(x.equals(y)); // true ? false? 原因は ?
    }
}
```

質問３：以下コンパイルを修正してください。

```
// Parent.java
public class Parent {
    public void hello() {
        System.out.println("parent method");
    }
}
// Child.java
public class Child extends Parent {
    @Override // 親クラスの強制的にOverrideする意味（アノテーション）
    public void hello() {
        System.out.println("child method");
    }
}
// Kicker.java
public class Kicker {
    public static void print(Child v) {
        v.hello();
    }
    public static void main(String...args) {
        print(new Parent());
        print(new Child());
    }
}
```

覚えておこう