作成者: mitsuya.85u@ray.ocn.ne.jp

4.1. 税金計算クラス（概念と実体、継承と多態）

抽象化・継承・多態・インターフェースの実験を行います。
税金計算という概念を実現するために税金計算の振舞いをインターフェースとして定義します。

以下のプログラムのMainクラスを見てください。
ここでは、AddShouhiZeiのインスタンスとAddJyuminZeiのインスタンスを、 TaxCalcInterface型の配列に押し込んでいます。そしてforループで回してcalculateメソッドを呼び出しています。
この部分、tc[i]に格納されているのがどのクラスのインスタンスなのかを調べていないことに注目してください。
これは、ポリモルフィズムの典型的な例です。これができるのは、TaxCalcInterfaceという共通の親クラスがあるからです。
TaxCalcInterfaceというインターフェースつまり抽象クラス(abstract class)を導入することで、ばらばらだったAddShouhiZeiとAddJyuminZeiがまとまり、 1つの「型(type)」を作ったという見方もできます。

【ソースコード】AddTax.java

```java
//抽象化・継承・多態・インターフェースの実験
//税金計算という概念を実現するために
//税金計算の振舞いをインターフェースとして定義する
interface TaxCalcInterface {
    void calculate();
    void show(int taxprice);
}
//①抽象クラスの宣言
abstract class TaxCalc implements TaxCalcInterface {
        int price ;
    //共通機能は抽象クラス内で実装
        public void show(int taxprice) {
            System.out.println(taxprice + "円です。");
        }
    //個々で異なる機能はメソッド宣言のみ
    public abstract void calculate( );
}
//②抽象クラスを継承
class AddShouhiZei extends TaxCalc {
        AddShouhiZei(int price) {
            this.price = price;
        }
    //抽象クラスで宣言されたcalculateメソッドの実装（オーバーライド）
        public void calculate( ){
            System.out.print("消費税率は5%のため");
            show((int)(price * 1.05));
        }
}
//③抽象クラスを継承
class AddJyuminZei extends TaxCalc {
        AddJyuminZei(int price) {
            this.price = price;
        }
    //抽象クラスで宣言されたcalculateメソッドの実装（オーバーライド）
        public void calculate( ){
            System.out.print("住民税率は20%のため");
            show((int)(price * 1.2));
        }
}
class AddTax {
        public static void main(String[ ] args) {
//④ポリモルフィズム
            TaxCalcInterface [] tc=new TaxCalcInterface [2];
            tc[0] = new AddShouhiZei (5000 );
            tc[1] = new AddJyuminZei (4000 );
            for (int i=0;i<tc.length;i++) {
                tc[i].calculate( );
            }
        }
}
```