SJC-P クラス、インタフェース、列挙型を宣言

インナークラス

■「通常」のインナークラス

以下のようなクラスではない

1. 静的ネストクラス
2. メソッドローカルなインナークラス
3. 無名インナークラス

・「通常」のインナークラスは、static 宣言を持つことはできない (1)
・Outer クラスのメンバーを参照できる (2)
・Outer クラスの参照は クラス名 .this で参照できる (3)
・Outer クラスのコード内からは、インナークラスを直接インスタンス化できる (4)
・Outer クラスのコード外から、インナークラスをインスタンス化するには、アウタークラスの参照が必要 (5)

```
package innerclass;

/**
 * 「通常」のインナークラス
 *  以下のようなクラスではない
 *   ・静的ネストクラス
 *   ・メソッドローカルなインナークラス
 *   ・無名インナークラス
 * @author Yagi Hiroto
 */
public class InnerClassTest1 {
    private String outerName = "InnerClassTest1";

    /**
     * 「通常」のインナークラス
     */
    class InnerClass {
        // (1) 「通常」のインナークラスは、static 宣言を持つことはできない
        public /* static */ void printName() {
            System.out.println("my name : InnerClass");
            // (2) Outer クラスのメンバーを参照できる
            System.out.println("outer name : " + outerName);
            // (3) Outer クラスの参照は クラス名 .this で参照できる
            System.out.println("outer  : " + InnerClassTest1.this);
        }
    }

    /*
     * Outer クラスのコード「内」から、インナークラスのインスタンスを生成する
     */
    public void printInnerName() {
        // (4) インナークラスを直接インスタンス化できる
        InnerClass ic = new InnerClass();
        ic.printName();
    }

    public static void main(String[] args) {
        InnerClassTest1 me = new InnerClassTest1();
        me.printInnerName();

        InnerClassTest2 ic2 = new InnerClassTest2();
        ic2.printOtherInnerName();
    }
}
class InnerClassTest2 {
    /*
     * Outer クラスのコード「外」から、インナークラスのインスタンスを生成する
     */
    public void printOtherInnerName() {
        // (5) インナークラスのインスタンス化には、アウタークラスの参照が必要
        InnerClassTest1.InnerClass ic
            = new InnerClassTest1().new InnerClass();
```

```
        ic.printName();
    }
}
```

■メソッドローカルなインナークラス

・Outer クラスのメンバにアクセス可能 (1)
・ローカル変数にはアクセスできない (2)
　・インナークラスはヒープに置かれるため、メソッドから抜けるとスタックにアクセスできなくなる
・final なローカル変数にはアクセスできる (3)
・メソッドを抜けても生存している (4)

```
package innerclass;

public class InnerClassTest3 {
    private int memberInt = 123;
    private Object obj;
    public void testMethodLocalInnerClass() {
        int       localInt = 456;
        final int finalInt = 789;

        class MethodInner {
            void printInt() {
                // (1) Outerクラスのメンバにアクセス可能
                System.out.println("Member :" + memberInt);
                // (2) ローカル変数にはアクセスできない
                //     インナークラスはヒープに置かれるため、メソッドから抜けるとスタックにアクセスで
きなくなる
                // System.out.println("Local  :" + localInt);
                // (3) final なローカル変数にはアクセスできる
                System.out.println("Final  :" + finalInt);
            }
            public String toString() {
                return "MethodInner";
            }
        }

        MethodInner mi = new MethodInner();
        mi.printInt();
        obj = mi;
    }

    public static void main(String[] args) {
        InnerClassTest3 me = new InnerClassTest3();
        me.testMethodLocalInnerClass();
        // (4) メソッドを抜けても生存している
        System.out.println(me.obj.toString());
    }
}
```

■無名インナークラス

・無名インナークラスとしてサブクラス化 (1)
・スーパークラスのメソッドをオーバーライド (2)
・末尾はセミコロン (3)

```
package innerclass;

public class InnerClassTest4 {

    public void testAnonymousInner() {
        /*
         * 無名インナークラス
         *  (1) 無名インナークラスとしてサブクラス化
         */
        BaseClass bc = new BaseClass() {
            // (2) スーパークラスのメソッドをオーバーライド
            public String getName() {
                return "AnonymousDerivClass";
            }
```

```
        }; // (3) 末尾はセミコロン

        System.out.println(bc.getName());
    }

    public static void main(String[] args) {
        (new InnerClassTest4()).testAnonymousInner();
    }
}
class BaseClass {
    public String getName() {
        return "BaseClass";
    }
}
```

■無名インナークラス (インターフェース実装クラス)

・無名インナークラスとしてインターフェースを実装 (1)
・インターフェースのメソッドを実装 (2)
・末尾はセミコロン (3)

```
package innerclass;

public class InnerClassTest5 {

    public void testAnonymousInner() {
        /*
         * 無名インナークラス
         *  (1) 無名インナークラスとしてインターフェースを実装
         */
        InfBase ib = new InfBase() {
            // (2) インターフェースのメソッドを実装
            public String getName() {
                return "AnonymousDerivClass";
            }
        }; // (3) 末尾はセミコロン

        System.out.println(ib.getName());
    }
    public static void main(String[] args) {
        (new InnerClassTest5()).testAnonymousInner();
    }
}
interface InfBase {
    String getName();
}
```

■無名インナークラス (メソッド引数として使用)

・セミコロン不要 (1)
・}); で終わる (2)

```
package innerclass;

public class InnerClassTest6 {
    public void testParamAnonymousClass(InfBase2 base) {
        System.out.println(base.getName());
    }
    public static void main(String[] args) {
        InnerClassTest6 me = new InnerClassTest6();

        me.testParamAnonymousClass(
                /* メソッド引数として無名インナークラスを利用する */
                new InfBase2() {
                    public String getName() {
                        return "Parameter AnonymousClass";
                    }
                } // (1) セミコロン不要
        );  // (2) }); で終わる
    }
}
interface InfBase2 {
    String getName();
```

}