2 학기
JAVA Class

‘이것이 자바다’

2025.11

Cover Description

made by Lewis

Test.1

- 아래와 같이 결과를 보여주도록 코딩 하시오.(for 문 2번만 사용 할 것)

```
2x1=2   3x1=3   4x1=4   5x1=5   6x1=6   7x1=7   8x1=8   9x1=9
2x2=4   3x2=6   4x2=8   5x2=10  6x2=12  7x2=14  8x2=16  9x2=18
2x3=6   3x3=9   4x3=12  5x3=15  6x3=18  7x3=21  8x3=24  9x3=27
2x4=8   3x4=12  4x4=16  5x4=20  6x4=24  7x4=28  8x4=32  9x4=36
2x5=10  3x5=15  4x5=20  5x5=25  6x5=30  7x5=35  8x5=40  9x5=45
2x6=12  3x6=18  4x6=24  5x6=30  6x6=36  7x6=42  8x6=48  9x6=54
2x7=14  3x7=21  4x7=28  5x7=35  6x7=42  7x7=49  8x7=56  9x7=63
2x8=16  3x8=24  4x8=32  5x8=40  6x8=48  7x8=56  8x8=64  9x8=72
2x9=18  3x9=27  4x9=36  5x9=45  6x9=54  7x9=63  8x9=72  9x9=81
```

Test.2

1. 1~100 숫자 중 3의 배수가 아닌 것을 모두 출력하고, 모두 몇 개인지 표시 하시오.
2. 1~100 숫자 중 7의 배수를 모두 출력하고, 모두 몇 개인지 표시 하시오.
3. 4 개의 정수를 입력 받아 큰 숫자부터 출력 하시오.
4. 2 수의 4칙 연산 계산기를 만드시오.

Test.3

- 길이가 3 인 배열을 생성하고 0 ~ 9 까지 중복 없는 값이 들어가도록 코딩 하시오. 단 첫번째 값은 0이 될 수 없음.

Test.4

- 야구게임을 완성하시오.

1. q, Q 면 게임 종료
2. Scanner 입력 받은 스트링 값 분해

```
int[] guess = new int[3];
for (int i = 0; i < 3; i++)
    guess[i] = input.charAt(i) - '0';
```

01 중첩 클래스

중첩 클래스란?

02 인스턴스 멤버 클래스

03 정적 멤버 클래스

04 로컬 클래스

05 바깥 멤버 접근

컴퓨터가 이해하는 코드는 누구라도 작성할 수 있습니다. 뛰어난 프로그래머는 사람이 이해하는 코드를 작성합니다.

중첩 클래스란?

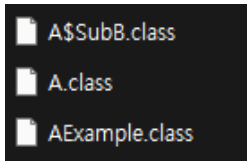
- 클래스 내부에 선언한 클래스
- 클래스의 멤버를 쉽게 사용할 수 있고 외부에는 중첩 관계 클래스를 감춤으로써 코드의 복잡성을 줄일 수 있다는 장점
- 중첩 클래스는 선언하는 위치에 따라
  클래스의 멤버로서 선언 - 멤버 클래스
  메소드 내부에서 선언 - 로컬 클래스

```
Class Member {
    [ static ] Class SubMember { … }
}
```

```
Class Member {
    void Method() {
        Class SubMember { … }
    }
}
```

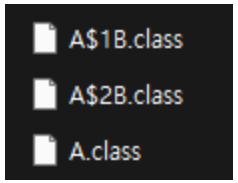
- 하나의 클래스이기 때문에 컴파일하면
  바이트코드파일(.class)이 별도로 생성

멤버 클래스 :



A $ B . class

로컬 클래스 :



A $1..n B .class

중첩 클래스

- 장점
  1. 논리적으로 그룹화
     내부 클래스와 외부 클래스를 함께 관리하는 것이 가능해 유지보수 면에서나 코드 이해성 면에서 편리해진다.
     내부 클래스로 인해 새로운 클래스를 생성하지 않아도 되므로 패키지를 간소화할 수 있다.
  2. 캡슐화 적용
     내부 클래스에 private 제어자를 적용, 캡슐화를 통해 클래스를 외부에 표출하지 않을 수 있다.
     클래스 구조를 숨김으로써 코드의 복잡성도 줄일 수 있다.
  3. 가독성 및 유지보수
     클래스를 따로 외부에 작성하는 경우보다 외부 클래스에 더 가깝게 위치하기때문에 시각적으로 읽기가 편해질 뿐 아니라 유지보수에 있어 이점을 가지게 된다.

인스턴스 멤버 클래스

```
Class MemberClass {
    [ public | private ] Class SubMemberClass { … }
}
```

1. 접근제한자 범위 설정
2. 내부 Class 임으로 Private 접근제한을 갖는 것이 일반적
3. MemberClass 가 있어야 SubMemberClass 를 생성할 수 있다.

```
MemberClass clsMem = new MemberClass();
MemberClass.SubMemberClass = clsMem.new SubMemberClass();
```

인스턴스 멤버 클래스

```java
public class A {
    //인스턴스 멤버 클래스
    class B {
        String sBStr = "";

        B() {
            this.sBStr = "Null";
        }

        B(String sBStr){
            this.sBStr = sBStr;
        }

        void ShowBstr() {
            System.out.println("sBStr : " + sBStr);
        }
    }

    //인스턴스 필드 값으로 B 객체 대입
    B field = new B();

    //생성자
    A() {
        B b = new B();
        b.ShowBstr();
    }

    //인스턴스 메소드
    void method() {
        B b = new B("method");
        b.ShowBstr();
    }
}
```

```java
public class AExample {
    public static void main(String[] args) {
        //A 객체 생성
        A a = new A();
        a.method();

        //B 객체 생성
        A.B b = a.new B();
        b.ShowBstr();
    }
}
```

인스턴스 멤버 클래스

```java
package ch09.sec02.exam02;

public class A {
    //인스턴스 멤버 클래스
    class B {
        //인스턴스 필드
        int field1 = 1;

        //정적 필드(Java17부터 허용)
        static int field2 = 2;

        //생성자
        B() {
            System.out.println("B-생성자 실행");
        }

        //인스턴스 메소드
        void method1() {
            System.out.println("B-method1 실행");
        }

        //정적 메소드(Java17부터 허용)
        static void method2() {
            System.out.println("B-method2 실행");
        }   //Java17부터 허용
    }

    //인스턴스 메소드
    void useB() {
        //B 객체 생성 및 인스턴스 필드 및 메소드 사용
        B b = new B();
        System.out.println(b.field1);
        b.method1();

        //B클래스의 정적 필드 및 메소드 사용
        System.out.println(B.field2);
        B.method2();
    }
}
```

```java
package ch09.sec02.exam02;

public class AExample {
    public static void main(String[] args) {
        //A 객체 생성
        A a = new A();

        //A 인스턴스 메소드 호출
        a.useB();
    }
}
```

```
B-생성자 실행
1
B-method1 실행
2
B-method2 실행
```

인스턴스 멤버 클래스

```java
public class Car {
    private String model;
    private Engine engine;

    public Car(String model, int horsePower) {
        this.model = model;
        this.engine = new Engine(horsePower);
    }

    public void start() {
        System.out.println(model + " 시동을 겁니다.");
        engine.run();
    }

    public void stop() {
        System.out.println(model + " 시동을 끕니다.");
    }

    private class Engine {
        private int horsePower;

        public Engine(int horsePower) {
            this.horsePower = horsePower;
        }

        public void run() {
            System.out.println("엔진이 가동 중입니다! (" + horsePower + "마력)");
        }
    }
}
```

```java
public class CarExecute {

    public static void main(String[] args) {
        Car car1 = new Car("소나타", 150);
        Car car2 = new Car("아반떼", 120);

        car1.start();
        car1.stop();

        car2.start();
        car2.stop();
    }
}
```

인스턴스 멤버 클래스 ➔ 메소드로 변환(GETTER/SETTER)

```java
public class CarM {
    private String model;
    private int horsePower;

    public CarM(String model, int horsePower) {
        this.model = model;
        setHorsePower(horsePower);  // 내부 클래스 객체 생성
    }

    public void start() {
        System.out.println(model + " 시동을 겁니다.");
        run();
    }

    public void stop() {
        System.out.println(model + " 시동을 끕니다.");
    }

    public void run() {
        System.out.println("엔진이 가동 중입니다! (" + horsePower + "마력)");
    }

    public int getHorsePower() {
        return horsePower;
    }

    public void setHorsePower(int horsePower) {
        if (horsePower < 0) {
            System.out.println("마력은 0 보다 커야 합니다.");
            this.horsePower = 0;
            return;
        }
        this.horsePower = horsePower;
    }
}
```

```java
public class CarExecute {

    public static void main(String[] args) {
        CarM car1 = new CarM("소나타", -150);
        Car car2 = new Car("아반떼", 120);

        car1.start();
        car1.stop();

        car2.start();
        car2.stop();
    }
}
```

```
마력은 0 보다 커야 합니다.
소나타 시동을 겁니다.
엔진이 가동 중입니다! (0마력)
소나타 시동을 끕니다.
아반떼 시동을 겁니다.
엔진이 가동 중입니다! (120마력)
아반떼 시동을 끕니다.
```

정적 멤버 클래스

```
Class MemberClass {
    [ public | private ] static Class SubMemberClass { … }
}
```

1. static 키워드와 함께 MemberClass 클래스의 멤버로 선언된 SubMemberClass 클래스를 말한다.
2. 정적 멤버 클래스 SubMemberClass 는 MemberClass 클래스 내부에서 사용되기도 하지만, MemberClass 클래스 외부에서 MemberClass와 함께 사용되는 경우가 많기 때문에 주로 default 또는 public 접근 제한자를 가진다.
3. SubMemberClass 객체는 MemberClass 클래스 내부 어디든 객체를 생성할 수 있다.

정적 멤버 클래스

```java
package ch09.sec03.exam01;

public class A {
    //인스턴스 멤버 클래스
    static class B {}

    //인스턴스 필드 값으로 B 객체 대입
    B field1 = new B();

    //정적 필드 값으로 B 객체 대입
    static B field2 = new B();

    //생성자
    A() {
        B b = new B();
    }

    //인스턴스 메소드
    void method1() {
        B b = new B();
    }

    //정적 메소드
    static void method2() {
        B b = new B();
    }
}
```

```java
package ch09.sec03.exam01;

public class AExample {
    public static void main(String[] args) {
        //B 객체 생성
        A.B b = new A.B();
    }
}
```

정적 멤버 클래스

```java
package ch09.sec03.exam02;

public class A {
    //정적 멤버 클래스
    static class SubB {
        //인스턴스 필드
        int field1 = 1;

        //정적 필드(Java17부터 허용)
        static int field2 = 2;

        //생성자
        SubB() {
            System.out.println("B-생성자 실행");
        }

        //인스턴스 메소드
        void method1() {
            System.out.println("B-method1 실행");
        }

        //정적 메소드(Java17부터 허용)
        static void method2() {
            System.out.println("B-method2 실행");
        }   //Java17부터 허용
    }
}
```

```java
package ch09.sec03.exam02;

public class AExample {
    public static void main(String[] args) {
        //B 객체 생성 및 인스턴스 필드 및 메소드 사용
        A.SubB b = new A.SubB();
        System.out.println(b.field1);
        b.method1();

        //B 클래스의 정적 필드 및 메소드 사용
        System.out.println(A.SubB.field2);
        A.SubB.method2();
    }
}
```

```
B-생성자 실행
1
B-method1 실행
2
B-method2 실행
```

정적 멤버 클래스

```java
package Test;

public class Bank {
    private static String bankName = "우리은행";

    // 📦 정적 멤버 클래스 (Static Nested Class)
    public static class Account {
        private String owner;
        private int balance;

        public Account(String owner, int balance) {
            this.owner = owner;
            this.balance = balance;
        }

        public void deposit(int amount) {
            balance += amount;
            System.out.println(owner + "님 입금 완료! 잔액: " + balance + "원");
        }

        public void withdraw(int amount) {
            if (balance < amount) {
                System.out.println("⚠ 잔액 부족! 출금 실패");
            } else {
                balance -= amount;
                System.out.println(owner + "님 출금 완료! 잔액: " + balance + "원");
            }
        }

        public void showInfo() {
            System.out.println("은행명: " + bankName);
            System.out.println("예금주: " + owner);
            System.out.println("현재 잔액: " + balance + "원");
            System.out.println();
        }
    }
}
```

```java
package Test;

public class BankExample {

    public static void main(String[] args) {
        Bank.Account acc1 = new Bank.Account("홍길동", 10000);
        Bank.Account acc2 = new Bank.Account("이영희", 5000);

        acc1.showInfo();
        acc2.showInfo();

        acc1.deposit(3000);
        acc2.withdraw(2000);

        acc1.showInfo();
        acc2.showInfo();
    }

}
```

로컬 클래스

```
[public] class A {
    public A() {   // 생성자
        class B {} // 로컬 클래스
    }
    public void method() {  // 메소드
        class B {}              // 로컬 클래스
    }
}
```

1. 생성자 또는 메소드 내부에 선언된 클래스를 말한다.
2. 로컬 클래스는 생성자와 메소드가 실행될 동안에만 객체를 생성할 수 있다

로컬 클래스

```java
package ch09.sec04.exam02;

public class A {
    //메소드
    void useB() {
        //로컬 클래스
        class B {
            //인스턴스 필드
            int field1 = 1;

            //정적 필드(Java17부터 허용)
            static int field2 = 2;

            //생성자
            B() {
                System.out.println("B-생성자 실행");
            }

            //인스턴스 메소드
            void method1() {
                System.out.println("B-method1 실행");
            }

            //정적 메소드(Java17부터 허용)
            static void method2() {
                System.out.println("B-method2 실행");
            }
        }

        //로컬 객체 생성
        B b = new B();

        //로컬 객체의 인스턴스 필드와 메소드 사용
        System.out.println(b.field1);
        b.method1();

        //로컬 클래스의 정적 필드와 메소드 사용(Java17부터 허용)
        System.out.println(B.field2);
        B.method2();
    }
}
```

```java
package ch09.sec04.exam02;

public class AExample {
    public static void main(String[] args) {
        //A 객체 생성
        A a = new A();

        //A 메소드 호출
        a.useB();
    }
}
```

로컬 클래스

```
package Test;

public class Calculator {
    public void calculate(int a, int b) {
        System.out.println("== 계산 시작 ==");

        final String operator = "덧셈";

        class Operation {
            public void add() {
                System.out.println(operator + " 결과: " + (a + b));
            }

            public void subtract() {
                System.out.println("뺄셈 결과: " + (a - b));
            }

            public void multiply() {
                System.out.println("곱셈 결과: " + (a * b));
            }

            public void divide() {
                if (b != 0)
                    System.out.println("나눗셈 결과: " + ((double) a / b));
                else
                    System.out.println("⚠ 0으로 나눌 수 없습니다!");
            }
        }
        Operation op = new Operation();
        op.add();
        op.subtract();
        op.multiply();
        op.divide();

        System.out.println("== 계산 종료 ==");
    }

    public static void main(String[] args) {
        Calculator calc = new Calculator();
        calc.calculate(10, 3);
    }

}
```

```
== 계산 시작 ==
덧셈 결과: 13
뺄셈 결과: 7
곱셈 결과: 30
나눗셈 결과: 3.3333333333333335
== 계산 종료 ==
```

Calculator 계산기 코딩 Test

Operation, Calculator class 를 생성하시오.

+, -, *, / 연산을 수행하는 계산기 코드를 생성하시오.

바깥 클래스의 객체 접근

바깥 클래스이름.this ➔ 바깥객체

```java
package ch09.sec05.exam02;

public class A {
    //A 인스턴스 필드
    String field = "A-field";

    //A 인스턴스 메소드
    void method() {
        System.out.println("A-method");
    }

    //인스턴스 멤버 클래스
    class B {
        //B 인스턴스 필드
        String field = "B-field";

        //B 인스턴스 메소드
        void method() {
            System.out.println("B-method");
        }

        //B 인스턴스 메소드
        void print() {
            //B 객체의 필드와 메소드 사용
            System.out.println(this.field);
            this.method();

            //A 객체의 필드와 메소드 사용
            System.out.println(A.this.field);
            A.this.method();
        }
    }

    //A의 인스턴스 메소드
    void useB() {
        B b = new B();
        b.print();
    }
}
```

```java
package ch09.sec05.exam02;

public class AExample {
    public static void main(String[] args) {
        //A 객체 생성
        A a = new A();

        //A 메소드 호출
        a.useB();
    }
}
```

```
B-field
B-method
A-field
A-method
```

중첩 클래스와 메소드 비교 1

1. 정의와 역할

	중첩 클래스 (Nested Class)	메서드 (Method)
정의	클래스 내부에 선언된 또 다른 클래스.	클래스 내에서 특정 작업을 수행하는 코드 블록.
역할	외부 클래스와 논리적으로 관련된 기능 그룹화. 독립적인 객체로 동작 가능.	특정 작업(기능)을 수행하고 값을 반환하거나 프로세스를 처리.

2. 사용 목적

	중첩 클래스 (Nested Class)	메서드 (Method)
목적	복잡한 데이터를 캡슐화하거나 외부 클래스와 관련된 기능을 모듈화.	단일 작업을 수행하거나 로직을 실행.
연관성	외부 클래스와 밀접한 연관이 있는 데이터 구조나 기능 정의.	외부 클래스의 데이터를 조작하거나 동작을 구현.

중첩 클래스와 메소드 비교 2

3. 구조적 차이

	중첩 클래스 (Nested Class)	메서드 (Method)
구조	클래스 정의 내에서 또 다른 클래스 정의.	반환 타입, 메서드 이름, 매개변수, 코드 블록으로 구성.

4. 독립성

	중첩 클래스 (Nested Class)	메서드 (Method)
독립적 객체	중첩 클래스는 외부 클래스의 객체와 별도로 객체를 생성할 수 있음.	메서드는 독립적으로 실행되지 않고 외부 클래스나 객체와 결합.

중첩 클래스와 메소드 비교 3

5. 캡슐화 및 재사용성

	중첩 클래스 (Nested Class)	메서드 (Method)
캡슐화	외부 클래스와 관련된 데이터와 동작을 캡슐화하여 구조적 그룹화 가능.	로직과 데이터를 캡슐화하여 외부에서 호출 가능.
재사용성	클래스 단위로 재사용할 수 있음.	메서드는 해당 클래스 내부에서 로직의 재사용성을 제공.

6. 메모리 및 성능

	중첩 클래스 (Nested Class)	메서드 (Method)
메모리	객체를 생성하면 메모리를 사용. 독립적으로 로드 가능.	호출 시 스택 메모리에 로드되며, 실행 종료 후 메모리 해제.
성능	클래스의 로드 시간과 객체 생성에 더 많은 리소스를 소모할 수 있음.	상대적으로 가벼운 실행.