09-1. 중첩 클래스와 중첩 인터페이스 소개

혼자 공부하는 자바 (신용권 저)

목차

시작하기 전에

[핵심 키워드] : 중첩 클래스, 멤버 클래스, 로컬 클래스, 중첩 인터페이스

[핵심 포인트]

객체 지향 프로그래밍에서 클래스들은 서로 긴밀한 관계를 맺고 상호작용을 한다. 그 중 특정한 클래스와 관계를 맺을 경우에는 클래스 내부에 선언하는 것이 좋다. 중첩 클래스와 중첩 인터페이스에 대해 알아본다.

시작하기 전에

❖ 중첩 클래스 (nested class)

- 클래스 내부에 선언한 클래스
- 두 클래스의 멤버들을 서로 쉽게 접근하게 하고, 외부에는 불필요한 관계 클래스 감춤
- 코드 복잡성 줄임

```
class ClassName {
  class NestedClassName {
  }
}
```

← 중첩 클래스

❖ 중첩 인터페이스 (nested interface)

- 인터페이스 역시 클래스 내부에 선언 가능
- 해당 클래스와 긴밀한 관계 갖는 구현 클래스 만들기 위함

```
class ClassName {
  interface NestedInterfaceName {
  }
}
```

← 중첩 인터페이스

중첩 클래스

❖ 멤버 클래스

- 클래스의 멤버로서 선언되는 중첩 클래스

❖ 로컬 클래스

- 메소드 내부에서 선언되는 중첩 클래스
- 메소드 실행할 때만 사용되고 메소드 종료되면 사라짐

선언 위치에 따른 분류		선언 위치	설명

멤버 클래스	인스턴스 멤버 클래스	`class A {`
		`  class B { … }`
		`}`
			A 객체를 생성해야만 사용할 수 있는 B 클래스

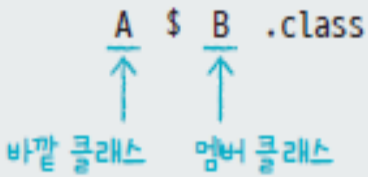
	정적 멤버 클래스	`class A {`
		`  static class B { … }`
		`}`
			A 클래스로 바로 접근할 수 있는 B 클래스

로컬 클래스		`class A {`
		`  void method() {`
		`  class B { … }`
		`  }`
		`}`
			method()가 실행할 때만 사용할 수 있는 B 클래스

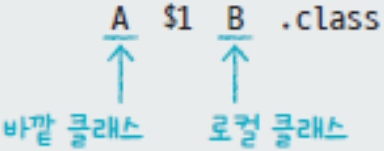
중첩 클래스

❖ 중첩 클래스를 컴파일하면 바이트 코드 파일(.class)이 별도로 생성

- 멤버 클래스 경우



- 로컬 클래스 경우



중첩 클래스

❖ 인스턴스 멤버 클래스

- static 키워드 없이 중첩 선언된 클래스
- 인스턴스 필드와 메소드만 선언 가능하고 정적 필드와 메소드는 선언할 수 없음

```
class A {
  /**인스턴스 멤버 클래스**/
  class B {
    B() { } ←———————————— 생성자
    int field1; ←———————————— 인스턴스 필드
    //static int field2; ←———————————— 정적 필드 (x)
    void method1() { } ←———————————— 인스턴스 메소드
    //static void method2() { } ←———————————— 정적 메소드 (x)
  }
}
```

중첩 클래스

- A 클래스 외부에서 B 객체 생성하려면 먼저 A 객체 생성 후 B 객체 생성 필요

A 클래스 외부

```
A  a = new A();
A.B  b = a.new B();
b.field1 = 3;
b.method1();
```

A 클래스 내부

```
class A {
  class B { … }

  void methodA() {
    B b = new B();
    b.field = 3;
    b.method1();
  }
}
```

중첩 클래스

❖ 정적 멤버 클래스

- static 키워드로 선언된 클래스
- 모든 종류의 필드와 메소드 선언 가능

```
class A {
  /**정적 멤버 클래스**/
  static class C {
    C() { }                    ← 생성자
    int field1;                ← 인스턴스 필드
    static int field2;         ← 정적 필드
    void method1() { }         ← 인스턴스 메소드
    static void method2() { }  ← 정적 메소드
  }
}
```

중첩 클래스

- A 클래스 외부에서 정적 멤버 클래스 C 객체 생성할 경우 A 객체 생성 필요하지 않음

```
A.C c = new A.C();
c.field1 = 3;     //인스턴스 필드 사용
c.method1();      //인스턴스 메소드 호출
A.C.field2 = 3;   //정적 필드 사용
A.C.method2();    //정적 메소드 호출
```

중첩 클래스

❖ 로컬 클래스

- 중첩 클래스를 메소드 내에서 선언할 수 있음
- 접근 제한자 및 static 붙일 수 없음
- 인스턴스 필드와 메소드만 선언할 수 있고 정적 필드와 메소드는 선언 불가

```
void method() {
  /**로컬 클래스**/
  class D {
    D() { }  <-- 생성자
    int field1;  <-- 인스턴스 필드
    //static int field2;  <-- 정적 필드 (x)
    void method1() { }  <-- 인스턴스 메소드
    //static void method2() { }  <-- 정적 메소드 (x)
  }
  D d = new D();
  d.field1 = 3;
  d.method1();
}
```

중첩 클래스

❖ 예시 – 중첩 클래스

```
01  package sec01.exam01;
02
03  /**바깥 클래스**/
04  class A {
05    A() { System.out.println("A 객체가 생성됨"); }
06
07    /**인스턴스 멤버 클래스**/
08    class B {
09      B() { System.out.println("B 객체가 생성됨"); }
10      int field1;
11      //static int field2;
12      void method1() { }
13      //static void method2() { }
14    }
15
16    /**정적 멤버 클래스**/
17    static class C {
18      C() { System.out.println("C 객체가 생성됨"); }
19      int field1;
20      static int field2;
21      void method1() { }
22      static void method2() { }
23    }
```

중첩 클래스

```
24
25      void method() {
26        /**로컬 클래스**/
27        class D {
28          D() { System.out.println("D 객체가 생성됨"); }
29          int field1;
30          //static int field2;
31          void method1() { }
32          //static void method2() { }
33        }
34        D d = new D();
35        d.field1 = 3;
36        d.method1();
37      }
38    }
```

중첩 클래스

❖ 예시 – 중첩 클래스의 객체 생성

```
01  package sec01.exam01;
02
03  public class Main {
04    public static void main(String[] args) {
05      A a = new A();
06
07      //인스턴스 멤버 클래스 객체 생성
08      A.B b = a.new B();
09      b.field1 = 3;
10      b.method1();
11
12      //정적 멤버 클래스 객체 생성
13      A.C c = new A.C();
14      c.field1 = 3;
15      c.method1();
16      A.C.field2 = 3;
17      A.C.method2();
18
19      //로컬 클래스 객체 생성을 위한 메소드 호출
20      a.method();
21    }
22  }
```

실행결과

```
A 객체가 생성됨
B 객체가 생성됨
C 객체가 생성됨
D 객체가 생성됨
```

중첩 클래스의 접근 제한

❖ 바깥 필드와 메소드에서 사용 제한

- 바깥 클래스에서 인스턴스 멤버 클래스 사용하는 경우

```
01  package sec01.exam02;
02
03  public class A {
04    //인스턴스 필드
05    B field1 = new B();
06    C field2 = new C();
07
08    //인스턴스 메소드
09    void method1() {
10      B var1 = new B();
11      C var2 = new C();
12    }
13
14    //정적 필드 초기화
15    //static B field3 = new B();
16    static C field4 = new C();
```

(o)

(o)

(o)

중첩 클래스의 접근 제한

```
17
18      //정적 메소드
19      static void method2() {
20        //B var1 = new B();  <-------- (x)
21        C var2 = new C();  <-------- (o)
22      }
23
24      //인스턴스 멤버 클래스
25      class B {}
26
27      //정적 멤버 클래스
28      static class C {}
29    }
```

중첩 클래스의 접근 제한

❖ 멤버 클래스에서 사용 제한

- 멤버 클래스 내부에서 바깥 클래스의 필드와 메소드에 접근하는 경우

```
class A {
  int field1;
  void method1() {…}

  static int field2;
  static void method2() {…}

  class B {
    void method() {
      field1 = 10;
      method1();

      field2 = 10;
      method2();
    }
  }
}
```

```
class A {
  int field1;
  void method1() {…}

  static int field2;
  static void method2() {…}

  static class C {
    void method() {
      field1 = 10;
      method1();

      field2 = 10;
      method2();
    }
  }
}
```

중첩 클래스의 접근 제한

```
01  package sec01.exam03;
02
03  public class A {
04    int field1;
05    void method1() { }
06
07    static int field2;
08    static void method2() { }
09
10    class B {
11      void method() {
```

중첩 클래스의 접근 제한

```
12          field1 = 10;
13          method1();
14
15          field2 = 10;
16          method2();
17        }
18      }
19
20      static class C {
21        void method() {
22          //field1 = 10;
23          //method1();
24
25          field2 = 10;
26          method2();
27        }
28      }
29    }
```

← 모든 필드와 메소드에 접근할 수 있음

← 인스턴스 필드와 메소드는 접근할 수 없음

중첩 클래스의 접근 제한

❖ 로컬 클래스에서 사용 제한

- 메소드의 매개 변수나 로컬 변수를 로컬 클래스에서 사용할 때의 제한
- 메소드가 종료되어도 계속 실행 상태로 존재하는 로컬 스레드 객체의 경우 등

- 매개 변수나 로컬 변수를 final 키워드로 선언해야 함
- 자바 8부터는 final 선언 하지 않아도 final 특성 부여되어 있음

```
01  package sec01.exam04;
02
03  public class Outter {
04    //자바 7 이전
05    public void method1(final int arg) {
06      final int localVariable = 1;
07      //arg = 100;
08      //localVariable = 100;
09      class Inner {
10        public void method() {
11          int result = arg + localVariable;
```

(x)

중첩 클래스의 접근 제한

```
12          }
13        }
14      }
15
16      //자바 8 이후
17      public void method2(int arg) {
18        int localVariable = 1;
19        //arg = 100;
20        //localVariable = 100;
21        class Inner {
22          public void method() {
23            int result = arg + localVariable;
24          }
25        }
26      }
27    }
```

(x)

중첩 클래스의 접근 제한

❖ 중첩 클래스에서 바깥 클래스 참조 얻기

- 바깥 클래스의 이름을 this 앞에 붙임

```
바깥클래스.this.필드
바깥클래스.this.메소드();
```

```
01  package sec01.exam05;
02
03  public class Outter {
04    String field = "Outter-field";
05    void method() {
06      System.out.println("Outter-method");
07    }
08
```

중첩 클래스의 접근 제한

```
09      class Nested {
10        String field = "Nested-field";
11        void method() {
12          System.out.println("Nested-method");
13        }
14        void print() {
15          System.out.println(this.field);
16          this.method();
17          System.out.println(Outter.this.field);
18          Outter.this.method();
19        }
20      }
21    }
```

15, 16 ← 중첩 객체 참조

17, 18 ← 바깥 객체 참조

중첩 클래스의 접근 제한

```
01  package sec01.exam05;
02
03  public class OutterExample {
04    public static void main(String[] args) {
05      Outter outter = new Outter();
06      Outter.Nested nested = outter.new Nested();
07      nested.print();
08    }
09  }
```

실행결과

```
Nested-field
Nested-method
Outter-field
Outter-method
```

중첩 인터페이스

❖ 중첩 인터페이스

- 클래스의 멤버로 선언된 인터페이스
- 해당 클래스와 긴밀한 관계 맺는 구현 클래스 만들기 위함

```
class A {
  [static] interface I {
    void method();
  }
}
```

중첩 인터페이스

- 인스턴스 멤버 인터페이스와 정적 멤버 인터페이스 모두 가능함

중첩 인터페이스

❖ 예시 – 중첩 인터페이스

```
01  package sec01.exam06;
02
03  public class Button {
04    OnClickListener listener;  <-- 인터페이스 타입 필드
05
06    void setOnClickListener(OnClickListener listener) {
07      this.listener = listener;
08    }                                  <-- 매개 변수의 다형성
09
10    void touch() {
11      listener.onClick();  <-- 구현 객체의 onClick() 메소드 호출
12    }
13
14    static interface OnClickListener {
15      void onClick();
16    }                                  <-- 중첩 인터페이스
17  }
```

중첩 인터페이스

❖ 예시 – 구현 클래스

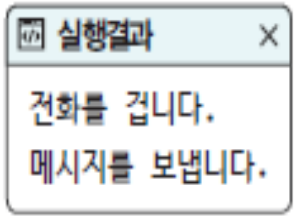
```
01  package sec01.exam06;
02
03  public class CallListener implements Button.OnClickListener {
04    @Override
05    public void onClick() {
06      System.out.println("전화를 겁니다.");
07    }
08  }
```

```
01  package sec01.exam06;
02
03  public class MessageListener implements Button.OnClickListener {
04    @Override
05    public void onClick() {
06      System.out.println("메시지를 보냅니다.");
07    }
08  }
```

중첩 인터페이스

❖ 버튼 이벤트 처리

```
01  package sec01.exam06;
02
03  public class ButtonExample {
04    public static void main(String[] args) {
05      Button btn = new Button();
06
07      btn.setOnClickListener( new CallListener() );
08      btn.touch();
09
10      btn.setOnClickListener( new MessageListener() );
11      btn.touch();
12    }
13  }
```



키워드로 끝내는 핵심 포인트

- 중첩 클래스: 클래스 내부에 선언한 클래스. 두 클래스의 멤버들을 서로 쉽게 접근하게 하고 외부에는 불필요한 관계 클래스 감추어 코드의 복잡성 줄임

- 멤버 클래스 : 클래스의 멤버로서 선언되는 중첩 클래스. 멤버 클래스는 바깥 객체의 필요 여부에 따라 인스턴스 멤버 클래스와 정적 멤버 클래스로 구분

- 로컬 클래스 : 생성자 또는 메소드 블록 내부에 선언된 중첩 클래스

- 중첩 인터페이스 : 클래스의 멤버로 선언된 인터페이스. 인스턴스 멤버 인터페이스와 정적 멤버 인터페이스 모두 가능함. 주로 정적 멤버 인터페이스를 UI 프로그래밍에서 이벤트 처리 목적으로 자주 활용함.

확인문제

❖ 중첩 멤버 클래스에 대한 설명으로 맞는 것에 O, 틀린 것에 X 하세요

- 인스턴스 멤버 클래스는 바깥 클래스의 객체가 있어야 사용될 수 있다 ( )
- 정적 멤버 클래스는 바깥 클래스의 객체가 없어도 사용될 수 있다 ( )
- 인스턴스 멤버 클래스 내부에는 바깥 클래스의 필드와 메소드를 사용할 수 있다 ( )
- 정적 멤버 클래스 내부에는 바깥 클래스의 인스턴스 필드를 사용할 수 있다 ( )

❖ 다음과 같이 Car 클래스 내부에 Tire와 Engine이 멤버 클래스로 선언되어 있습니다. 바깥 클래스에서 멤버 클래스의 객체를 생성하는 코드를 빈 칸에 작성해 보세요.

소스 코드 Car.java

```
01  package sec01.verify.exam03;
02
03  public class Car {
04    class Tire { }
05    static class Engine { }
06  }
```

확인문제

소스 코드 NestedClassExample.java

```
01  package sec01.verify.exam03;
02
03  public class NestedClassExample {
04    public static void main(String[] args) {
05      Car myCar = new Car();
06
07      Car.Tire tire = 
08
09      Car.Engine engine = 
10    }
11  }
```

확인문제

❖ 다음 Chatting 클래스에서 컴파일 에러가 발생하는 이유는 무엇입니까?

소스 코드 Chatting.java

```
01  package sec01.verify.exam04;
02
03  public class Chatting {
04    void startChat(String chatId) {
05      String nickName = null;
06      nickName = chatId;
07
08      class Chat {
09        public void start() {
10          while (true) {
11            String inputData = "안녕하세요";
12            String message = "[" + nickName + "] " + inputData;
13            sendMessage(message);
14          }
15        }
16
17        void sendMessage(String message) {
18        }
19      }
20
21      Chat chat = new Chat();
22      chat.start();
23    }
24  }
```


혼자 공부하는 자바
혼자 공부하는 자바
1:1 과외처럼 공부하고 싶을 때
작심삼일, 하다가 포기하게 될 때
어제 배운 내용이 생각나지 않을 때
신용권 지음
JAVA 8&11 지원
245개의 손코딩 예제
무료 동영상 강의
별책 용어 노트
한빛미디어

Thank You!