9. 내부 클래스

이 장에서는 클래스 블록 안에 선언되는 내부 클래스에 대해 설명하기로 하겠습니다. 자바에서는 클래스가 다른 클래스를 얼마든지 포함할 수 있습니다. 클래스 안에 멤버 메서드와 같은 레벨의 클래스를 가질 수 있으며, 메서드 안에도 클래스가 선언될 수 있습니다. 내부클래스의 쓰임새를 정확하게 이해한다면 여러분은 고급 프로그래머에 한발 더 다가서게 됩니다.

9장의 주요 내용입니다.

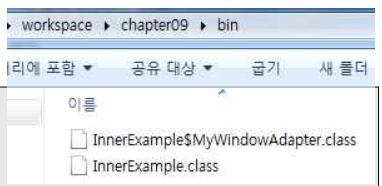
- 멤버 내부 클래스
- 지역 내부 클래스
- 익명 내부 클래스
- 내부 클래스 객체 생성
- 변수 참조
- static inner

9.1. 내부 클래스(Inner class)

내부 클래스의 개념은 JDK 1.1에서 포함되었으며 클래스 안에 선언한 클래스이기 때문에 내포(nested) 클래스라고도 합니다. 내부 클래스는 존재 위치와 클래스 이름의 유/무에 따라 4가지로 나누어 설명 할 수 있습니다. 클래스 안에 멤버로 존재하는 멤버 이너 클래스(Member Inner Class), 메서드 또는 블록 안에 존재하며 이름이 있는 로컬 이너 클래스(Local Inner Class), 그리고 메서드 또는 블록 안에 존재하지만 이름이 없는 익명 이너 클래스(Anonymous Inner Class)와 클래스 선언과 동시에 변수 선언을 한 익명 클래스 변수(Anonymous Class Variable)가 있습니다.

9.1.1. 멤버 내부 클래스(Member Inner Class)

클래스의 멤버로 필드나 메서드가 아닌 클래스가 올 수 있는데, 이를 멤버 내부(Inner) 클래스라고 합니다. 필드나 메서드처럼 public, private, protected, static, final 등의 접근 제한자가 올 수 있으며, 클래스 선언부를 주 클래스 안으로 옮겨 놓으면 됩니다. 주 클래스를 컴파일하면 자동으로 내부클래스도 컴파일하게 되며, 주 클래스 이름과 내부클래스 이름을 "$" 문자로 연결한 클래스 파일이 생성됩니다. 코드를 컴파일하면 클래스인 InnerExample.class파일과 내부 클래스인 InnerExample$MyWindowAdapter.class파일이 생성됩니다.

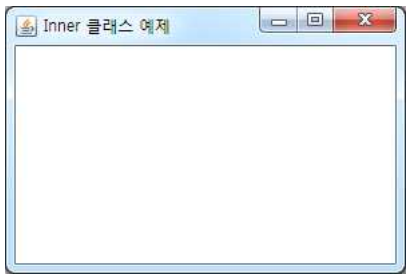


InnerExample.java

```
1:  import java.awt.*;
2:  import java.awt.event.*;
3:
4:  public class InnerExample {
5:     private Frame f;
6:
5:     public InnerExample() {
6:        f = new Frame("Inner 클래스 예제");
7:     }
8:     public void launchFrame() {
9:        f.addWindowListener( new MyWindowAdapter() );
10:       f.setSize(300, 200);
11:       f.setVisible(true);
12:    }
13:    public static void main (String[] args) {
14:       InnerExample ie = new InnerExample();
15:       ie.launchFrame();
16:    }
17:    private class MyWindowAdapter extends WindowAdapter {
18:       public void windowClosing(WindowEvent e) {
19:          System.exit(0);
```

```
20:         }
21:     }//end inner class
22: }//end class
```

예제를 실행시키면 제목 표시줄이 “Inner 클래스 예제”인 윈도우 창이 하나 실행됩니다. 그리고 종료버튼을 누르면 윈도우 창이 닫히는 예입니다. 이 예제에서는 윈도우 종료 이벤트 처리를 위한 부분을 내부 클래스로 구현한 것입니다.



9.1.2. 지역 내부 클래스(Local Inner Class)

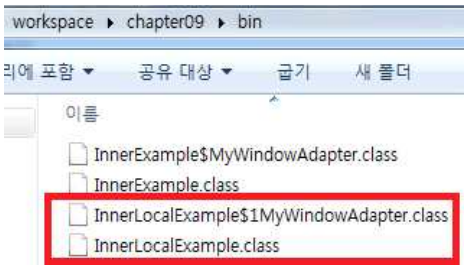
로컬 이너 클래스는 메서드 또는 선언된 블록 내에서만 유효합니다. 클래스의 영역 범위가 블록의 영역 범위와 같으므로 접근제한자가 필요 없습니다. 컴파일하면 주 클래스 이름과 블록번호(블록이 선언된 순서), 그리고 내부 클래스 이름을 "$" 문자로 연결한 파일을 생성합니다.

다음 코드는 메서드 내에서 이름을 부여하여 클래스를 정의한 것입니다.

InnerLocalExample.java

```
1:  import java.awt.*;
2:  import java.awt.event.*;
3:
4:  public class InnerLocalExample {
5:     private Frame f;
6:
7:     public InnerLocalExample() {
8:        f = new Frame("Local Class 예제");
9:     }
10:
11:    public void launchFrame() {
12:       class MyWindowAdapter extends WindowAdapter {
13:          public void windowClosing(WindowEvent we) {
14:             System.exit(0);
15:          }
16:       }
17:       f.addWindowListener( new MyWindowAdapter() );
18:       f.setSize(300, 200);
19:       f.setVisible(true);
20:    }
21:
22:    public static void main(String[] args) {
23:       InnerLocalExample ile = new InnerLocalExample();
24:       ile.launchFrame();
25:    }
26: }
```

이 예제는 InnerExample.java와 동일하게 동작하는 클래스입니다. 컴파일하면 내부 클래스인 InnerLocalExample$1MyWindowAdapter.class파일과 Top-Level 클래스인 InnerLocalExample.class파일을 생성합니다.



로컬 이너 클래스도 온전한 클래스이므로 다른 클래스를 상속 받을 수도 있으며 재정의 또는 다형성 기능도 사용할 수 있습니다. 앞의 소스코드에서 launchFrame()메서드 안에 선언한 MyWindowAdapter 클래스가 WindowAdapter 클래스를 상속받은 것을 알 수 있습니다.

로컬 이너 클래스는 외부 클래스의 모든 멤버를 참조할 수 있습니다. 그러나 로컬 이너클래스에서 지역변수를 참조하기 위해서는 지역변수가 final로 선언[34]되어 있어야 합니다. 왜냐하면 메서드가 종료된 후에 로컬 이너 클래스가 실행되는 경우도 있기 때문인데 이때는 지역 변수가 사라진 후이므로 변수의 값을 제대로 읽을 수 없기 때문입니다. 지역변수는 메서드가 끝나면 사라집니다. 그러나 로컬 클래스는 final 지역변수가 절대 바뀌지 않는다는 것을 알기 때문에 내부클래스에서도 final 지역변수를 그대로 저장해서 사용합니다. 여기서 로컬 이너클래스를 사용하는 이유는 호출되는 메서드의 인자 값에 따라 이벤트 처리를 다르게 해야 할 경우가 있기 때문입니다.

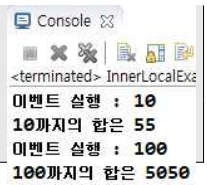
다음 소스코드에서 final 지역변수의 예를 보여주고 있습니다.

InnerLocalExample2.java

```
1:  public class InnerLocalExample2 {
2:
3:     public static void main(String[] args) {
4:        new InnerLocalExample2().doIt(10);
5:        new InnerLocalExample2().doIt(100);
6:     }
7:
8:     public void doIt(final int data) {
9:
10:       class EventHandler {
11:          void callBack() {
12:             int sum = 0;
13:             System.out.println("이벤트 실행 : " + data);
14:             for(int i=1; i<=data; i++) {
15:                sum = sum + i;
16:             }
17:             System.out.println(data +"까지의 합은 " + sum);
18:          }
```

34) 8장 final에서 설명했던 내용입니다. JDK 컴파일러 버전이 1.8 이상일 경우에는 컴파일러가 final을 붙여줍니다.

```
19:         }//end inner local class
20:
21:         new EventHandler().callBack();
22:     }//end doIt()
23: }//end class
```



예제에서 EventHandler 클래스는 실제 이벤트처리를 하는 콜백메서드를 구현하는 것이 아니므로 직접 객체를 생성 후 메서드를 호출하는 코드를 포함시켰습니다. 코드에서는 doIt() 메서드를 두 번 호출하고 있습니다. 호출할 때 인자의 값이 다름을 주의하세요. 이렇게 인자의 값이 다르게 동일한 메서드가 호출 되고, 호출되는 메서드 안에서 이벤트 처리를 해야 할 경우에는 로컬 이너클래스를 사용해야 합니다. 이렇게 하면 메서드 인자를 통해서 내부클래스에 사용될 값을 전달할 수 있습니다. 여기서 또 하나 주의해야 할 점은 메서드 파라미터 변수가 final로 선언되어야 한다는 것입니다. final 키워드 설명할 때 언급했었던 부분입니다. 로컬 이너클래스에서 지역변수(로컬변수)를 참조하기 위해서는 지역변수는 반드시 final로 선언되어야 합니다. 그러므로 로컬이너클래스에서 참조하는 파라미터 변수는 메서드 내에서 값이 수정될 수 없으며, 지역변수 또한 한번 할당되면 값을 수정할 수 없습니다.

9.1.3. 익명 클래스(Anonymous Inner Class)

익명 클래스(익명 이너클래스)는 블록 내부에 존재하는 지역 클래스에서 이름을 생략한 것입니다. 이름을 생략한 지역 클래스는 new 키워드로 객체를 생성하는 부분에서 메서드를 구현합니다. 컴파일하면 주 클래스 이름에 지역 클래스의 번호를 "$" 문자로 연결한 파일을 생성합니다.

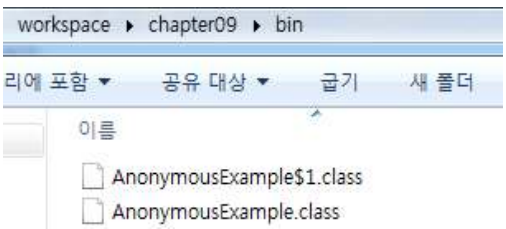
다음 코드는 앞에서 내부클래스와 지역클래스 예제와 동일하게 실행되는 프로그램을 익명 클래스를 이용하여 구현한 것입니다. 익명 클래스 구현부 코드의 형식에 주의하세요. 그리고 괄호가 닫히는 부분에 주의하세요.

AnonymousExample.java

```
1:  import java.awt.*;
2:  import java.awt.event.*;
3:
4:  public class AnonymousExample {
5:      private Frame f;
6:
7:      public AnonymousExample() {
8:          f = new Frame("Anonymous Class 예제");
9:      }
10:
11:     public void launchFrame() {
12:         f.addWindowListener( new WindowAdapter() {
```

```
13:            public void windowClosing(WindowEvent we) {
14:               System.exit(0);
15:            }
16:         } );
17:         f.setSize(300, 200);
18:         f.setVisible(true);
19:      }
20:      public static void main(String[] args) {
21:         AnonymousExample ae = new AnonymousExample();
22:         ae.launchFrame();
23:      }
24: }
```

코드를 컴파일하면 AnonymousExample.class 파일과 AnonymousExample$1.class 파일을 생성합니다. 실행했을 때 앞의 윈도우 창이 보이는 프로그램과 같은 창이 실행됩니다.



9.1.4. 익명 클래스 변수(Anonymous Class Variable)

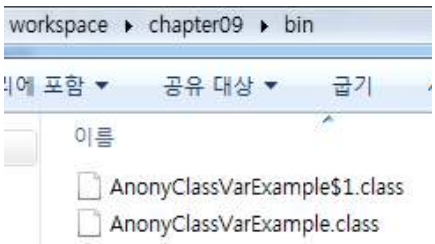
내부 클래스를 선언하면서 인스턴스 생성까지 할 수도 있습니다. 익명 클래스 변수는 멤버 변수로 존재할 수도 있으며, 지역변수로 존재할 수도 있습니다. 익명 클래스 변수를 사용하면 내부클래스의 이름을 일일이 지정할 필요도 없으며, 내부클래스 인스턴스를 별도로 선언할 필요도 없습니다. 나중에 안드로이드 프로그래밍을 공부할 예정이라면 반드시 알아두어야 할 코드입니다. 컴파일하면 주 클래스 이름에 지역 클래스의 번호를 "$" 문자로 연결한 파일을 생성합니다.

AnonyClassVarExample.java

```
1:  import java.awt.*;
2:  import java.awt.event.*;
3:
4:  public class AnonyClassVarExample {
5:     private Frame f;
6:
7:     public AnonyClassVarExample() {
8:        f = new Frame("Anonymous Class");
9:     }
10:
11:    WindowListener handler = new WindowAdapter() {
12:       public void windowClosing(WindowEvent we){
13:          System.exit(0);
14:       }
```

```
15:     };
16:
17:     public void launchFrame() {
18:        f.addWindowListener( handler );
19:        f.setSize(300, 200);
20:        f.setVisible(true);
21:     }
22:     public static void main(String[] args) {
23:        AnonyClassVarExample ae = new AnonyClassVarExample();
24:        ae.launchFrame();
25:     }
26: }
```

컴파일하면 다음처럼 클래스파일이 생성됩니다.



내부 클래스들에 관한 개념이 이해가 잘 되지 않는다면 예제 코드들의 형식을 주의 깊게 살펴보시기 바랍니다.

9.2. 내부 클래스(Inner class)의 사용

9.2.1. 내부클래스의 객체 참조

내부 클래스에서 외부 클래스 멤버를 참조할 때에 특별한 제약은 없습니다. 그러나 내부클래스를 참조하고자 할 경우에는 참조하는 위치가 어디냐에 따라 내부클래스의 객체 생성 방법이 달라집니다. 내부클래스의 멤버를 참조하는 방법을 예제 코드를 통해서 살펴보기로 하겠습니다.

다음 코드는 내부클래스를 포함하는 외부클래스 Outer1 클래스의 testTheInner() 메서드에서 Inner1 클래스를 참조하는 예입니다.

Outer1.java

```
1:  public class Outer1 {
2:     private int data;
3:
4:     public class Inner1 {
5:        public void doIt() {
6:           data++;
7:           System.out.println("Inner 클래스의 메서드 호출됨");
8:           System.out.println("data값은 " + data);
9:        }
10:    }
11:
12:    public void testTheInner() {
13:       Inner1 in = new Inner1();
14:       in.doIt();
15:    }
16:
17:    public static void main(String[] args) {
18:       Outer1 out = new Outer1();
19:       out.testTheInner();
20:    }
21: }
```

```
Console
<terminated> Outer1 [Java Applicatio
Inner 클래스의 메서드 호출됨
data값은 1
```

Inner1클래스와 testTheInner()메서드는 모두 Outer1이라는 동일한 클래스의 범위 안에 있습니다. tesetTheInner() 메서드에서 Inner1 내부클래스의 인스턴스를 생성할 때에는 보통의 클래스의 인스턴스를 생성하는 방법으로 생성하면 됩니다.

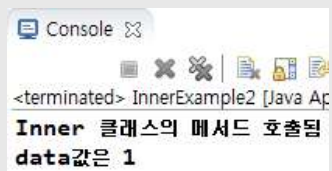
다음 코드는 내부클래스를 별도의 클래스에서 참조하는 예입니다.

Outer2.java

```
1:  public class Outer2 {
2:     private int data;
3:
4:     public class Inner2 {
5:        public void doIt() {
6:           data++;
7:           System.out.println("Inner 클래스의 메서드 호출됨");
8:           System.out.println("data값은 " + data);
9:        }
10:    }
11: }
```

InnerExample2.java

```
1:  public class InnerExample2 {
2:     public static void main(String[] args) {
3:
4:        Outer2 out = new Outer2();
5:        Outer2.Inner2 in = out.new Inner2();
6:        in.doIt();
7:     }
8:  }
```



예에서처럼 다른 클래스에서 내부 클래스를 참조하려면 외부클래스의 인스턴스를 이용해야 합니다. InnerExample2 클래스의 4,5라인은 다음과 같이 표현할 수 있습니다.

```
4:        Outer2.MyInner2 in = new Outer2().new MyInner2();
```

만일 내부클래스가 static으로 선언되어 있다면 다음과 같이 선언 및 참조해야 합니다. 내부클래스가 static으로 선언될 경우 내부클래스의 메서드들은 외부클래스의 멤버를 직접 참조 할 수 없음에 유의하세요.

```
4:        Outer2.MyInner2 in = new Outer2.MyInner2();
```

9.2.2. 내부클래스의 변수 참조

다음 프로그램은 내부클래스에서 변수를 참조하는 예를 보인 것입니다. 내부클래스를 포함한 클래스의 멤버변수와 내부클래스의 멤버변수 이름이 같으면 어느 변수를 참조하는지 구분해야 합니다.

내부클래스 메서드에서 변수 참조 시 우선순위는 내부클래스의 지역변수 -> 내부클래스의 멤버변수 -> 외부클래스의 멤버변수 순입니다.

Outer3.java

```
1:  public class Outer3 {
2:     private int data=10;
3:
4:     public class Inner3 {
5:        private int data=20;
6:
7:        public void doIt(int data) {
8:           System.out.println( data );
9:           System.out.println( this.data );
10:          System.out.println( Outer3.this.data );
11:       }
12:    }
13: }
```

InnerExample3.java

```
1:  public class InnerExample3 {
2:     public static void main(String[] args) {
3:        Outer3.Inner3 in = new Outer3().new Inner3();
4:        in.doIt(30);
5:     }
6:  }
```

```
Console
<terminated> Inne
30
20
10
```

다음 코드는 내부클래스가 메서드 안에 선언된 경우 변수 참조 예를 보입니다. 메서드 내에 클래스가 선언되면 로컬 이너 클래스(Local Inner Class) 즉 지역클래스라고 하였습니다. 지역클래스에서는 자신을 포함하는 메서드의 지역변수들 중에서 지역클래스 내부에 선언된 변수가 아니면 final 지역 변수를 제외한 어떤 지역 변수도 참조할 수 없습니다.

Outer4.java

```
1:  public class Outer4 {
2:     private int data=10;
3:
4:     public Object makeTheInner(final int localData) {
5:        final int FINAL_LOCAL_DATA=20;
6:
7:        class MyInner4 {
8:           public String toString() {
9:              return ( "data=" + data +
10:                "\nlocalData=" + localData +
11:                "\nFINAL_LOCAL_DATA=" + FINAL_LOCAL_DATA );
12:          }
13:       }//end local class
14:       return new MyInner4();
15:    }//end makeTheInner()
16:
17:    public static void main(String[] args) {
18:       Outer4 out = new Outer4();
```

```
Console
<terminated> Outer4 [Java Applic
data=10
localData=30
FINAL_LOCAL_DATA=20
```

```
19:         Object in = out.makeTheInner(30);
20:         System.out.println(in);
21:     }
22: }//end class
```

9.2.3. 내부클래스의 제한자 사용

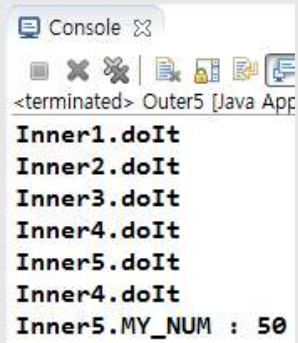
내부클래스는 모든 접근제한 모드를 사용할 수 있으며, static이나 abstract를 제한자로 선언하여 사용할 수도 있습니다.
내부 클래스는 static 멤버를 가질 수 없습니다. 그러나 static 멤버 선언이 필요할 경우에 내부 클래스를 static으로 선언할 수 있습니다. 내부클래스를 static으로 선언하면 Top-Level 클래스가 되어 static 멤버를 가질 수 있습니다. 즉, static 멤버를 갖는 내부클래스를 선언하려면 static으로 선언해야 합니다.

다음 코드는 내부클래스에서 접근 제한모드를 테스트하기 위한 예입니다.

Outer5.java

```
1:  public class Outer5 {
2:     public class Inner1 {
3:        public void doIt() {
4:           System.out.println("Inner1.doIt");
5:        }
6:     }
7:     protected class Inner2 {         //하위 클래스에서 참조 가능
8:        public void doIt() {
9:           System.out.println("Inner2.doIt");
10:       }
11:    }
12:    /* friendly */ class Inner3 {    //같은 패키지 내의 클래스에서 참조 가능
13:       public void doIt() {
14:          System.out.println("Inner3.doIt");
15:       }
16:    }
17:    private class Inner4 {           //클래스 안에서만 참조 가능
18:       public void doIt() {
19:          System.out.println("Inner4.doIt");
20:       }
21:    }
22:    public static class Inner5 {     //static 멤버를 갖는 클래스는 static으로 선언
23:       public static final int MY_NUM = 50;
24:       public static void doIt() {
25:          System.out.println("Inner5.doIt");
26:       }
27:    }
28:    public void go() {
29:       Inner1 in1 = new Inner1();
```

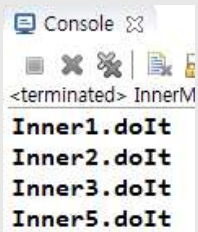
```
30:        in1.doIt();
31:        Inner2 in2 = new Inner2();
32:        in2.doIt();
33:        Inner3 in3 = new Inner3();
34:        in3.doIt();
35:        Inner4 in4 = new Inner4();
36:        in4.doIt();
37:        Inner5.doIt();
38:     }//end go()
39:
40:     public static void main(String[] args) {
41:        Outer5 out = new Outer5();
42:        out.go();
43:        Outer5.Inner4 in = out.new Inner4();
44:        in.doIt();
45:        System.out.println("Inner5.MY_NUM : " + Inner5.MY_NUM);
46:     }//end main()
47: }//end Outer5 class
```



다음은 다른 클래스에서 내부클래스의 접근제한자를 테스트할 수 있는 코드입니다.

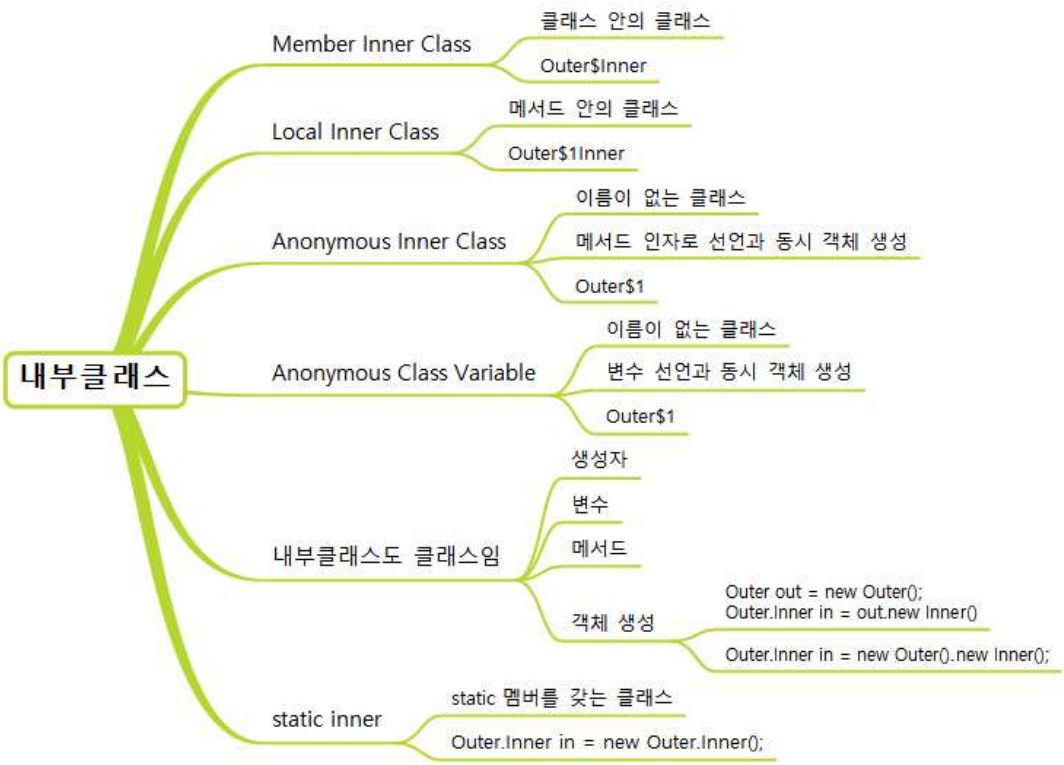
InnerModifierExample.java

```
1:  public class InnerModifierExample {
2:
3:     public static void main(String[] args) {
4:        Outer5.Inner1 in1 = new Outer5().new Inner1();
5:        in1.doIt();
6:        Outer5 out5 = new Outer5();
7:        Outer5.Inner2 in2 = out5.new Inner2();
8:        in2.doIt();
9:        Outer5.Inner3 in3 = out5.new Inner3();
10:       in3.doIt();
11: //    Outer5.Inner4 in4 = out5.new Inner4();//Outer5.Inner4 is not visible
12:       Outer5.Inner5.doIt();
13:    }
14: }//end Outer5 class
```



내부클래스에 제한자가 붙는다고 해서 특별히 달라질 것은 없습니다. 내부클래스의 제한자들 중에서 접근제한자는 메서드에 붙는 제한자와 동일한 기능을 갖는다고 생각하면 됩니다. 그리고 abstract는 내부클래스를 추상 클래스로 만들며, static은 내부클래스가 static 멤버를 갖게 할 경우 사용합니다.

9.3. 마인드맵 정리


내부클래스
Member Inner Class
클래스 안의 클래스
Outer$Inner
Local Inner Class
메서드 안의 클래스
Outer$1Inner
Anonymous Inner Class
이름이 없는 클래스
메서드 인자로 선언과 동시 객체 생성
Outer$1
Anonymous Class Variable
이름이 없는 클래스
변수 선언과 동시 객체 생성
Outer$1
내부클래스도 클래스임
생성자
변수
메서드
객체 생성
Outer out = new Outer();
Outer.Inner in = out.new Inner()
Outer.Inner in = new Outer().new Inner();
static inner
static 멤버를 갖는 클래스
Outer.Inner in = new Outer.Inner();