inner class와 lambda expression

2017-09-13

이승진

**학습목표**

inner class 문법을 배운다.

anonymous inner class 문법을 배운다.

lambda expression 문법을 배운다.

**목차**

[1. inner class 2](#_Toc493079624)

[1) inner class 란? 2](#_Toc493079625)

[2) 예제 2](#_Toc493079626)

[2. static inner class 5](#_Toc493079627)

[1) static inner class 란? 5](#_Toc493079628)

[2) 예제 5](#_Toc493079629)

[3. annoymous inner class 7](#_Toc493079630)

[1) 예제 #1 7](#_Toc493079631)

[2) 예제 #2 9](#_Toc493079632)

[3) 예제 #3 10](#_Toc493079633)

[4) 예제 #4 11](#_Toc493079634)

[5) 예제 #5 12](#_Toc493079635)

[6) 예제 #6 13](#_Toc493079636)

# inner class

## inner class 란?

어떤 클래스 내부에 정의된 클래스를 inner class라고 부른다.

|  |
| --- |
| class OutterClass {  class InnerClass {  }  } |

inner class 객체를 생성할 때, outter class 객체에 대한 this 가 inner class 객체에 자동으로 전달된다.

그래서 inner class의 메소드에서 outter class 객체에 대한 this 를 사용할 수 있다.

outter class 객체에 대한 this 가 있는 곳에서만 (instance method),

inner class 객체를 생성할 수 있다.

## 예제

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36 | package net.skhu.inner;  public class OutterClass1 {  int a;  class InnerClass {  int b;  void method2() {  b = 5;  this.b = 6;  a = 7;  OutterClass1.this.a = 8;  instanceMethod();  OutterClass1.this.instanceMethod();  // this.instanceMethod(); 컴파일 에러  main(null);  }  }  public void instanceMethod() {  InnerClass obj = new InnerClass();  a = 1;  this.a = 2;  obj.method2();  }  public static void main(String[] args) {  OutterClass o = new OutterClass();  o.instanceMethod();  // InnerClass obj = new InnerClass(); 컴파일 에러  // a = 3; 컴파일 에러  // this.a = 4; 컴파일 에러  }  } |

(줄11~12)

|  |
| --- |
| b = 5;  this.b = 6; |

위 두 줄은, 둘 다 this.b 멤버 변수에 값을 대입하는 코드이다.

b 는 InnerClass 의 멤버 변수이다.

this. 은 생략할 수 있다.

inner class 메소드에서 this 는, inner class 객체를 가르킨다.

(줄13~14)

|  |
| --- |
| a = 7;  OutterClass1.this.a = 8; |

위 두 줄은, 둘 다 OutterClass1.this.a 멤버 변수에 값을 대입하는 코드이다.

a 는 OutterClass1 의 멤버 변수이다.

OutterClass1.this. 은 생략할 수 있다.

inner class 메소드에서, outter class 객체에 대한 this를 사용할 수 있다.

(줄15~16)

|  |
| --- |
| instanceMethod();  OutterClass1.this.instanceMethod(); |

위 두 줄은, 둘 다 OutterClass1.this.instanceMethod() 메소드를 호출하는 코드이다.

instanceMethod() 는 OutterClass1 의 메소드이다.

OutterClass1.this. 은 생략할 수 있다.

inner class 메소드에서, outter class 객체의 메소드를 호출할 수 있다.

(줄17)

|  |
| --- |
| // this.instanceMethod(); 컴파일 에러 |

주석으로 막지 않았다면, 위 코드애서 컴파일 에러가 발생한다.

inner class 메소드에서 this는, 당연히 inner class 객체에 대한 this 이다.

inner class 에 instanceMethod() 메소드가 없으므로, 위 코드에서 컴파일 에러가 발생한다.

(줄18)

|  |
| --- |
| main(null); |

inner class 메소드에서, outter class 의 static 메소드를 호출할 수 있다.

(줄22~27)

|  |
| --- |
| public void instanceMethod() {  InnerClass obj = new InnerClass();  a = 1;  this.a = 2;  obj.method2();  } |

inner class 객체를 생성할 때, outter class 객체에 대한 this 가 전달되어야 한다.

instanceMethod() 메소드는 인스턴트 메소드이기 때문에, this 를 사용할 수 있다.

그래서 inner class 객체도 생성할 수 있다.

|  |
| --- |
| a = 1;  this.a = 2; |

위 두 줄은, 둘 다 this.a 멤버 변수에 값을 대입하는 코드이다.

a 는 OutterClass1 의 멤버 변수이다.

this는 OutterClass1 객체를 가르킨다.

this. 은 생략할 수 있다.

(줄29~33)

|  |
| --- |
| public static void main(String[] args) {  // InnerClass obj = new InnerClass(); 컴파일 에러  // a = 3; 컴파일 에러  // this.a = 4; 컴파일 에러  } |

main 메소드는 static 메소드이다.

static 메소드에서는 this를 사용할 수 없다.

따라서 inner class 객체를 생성할 수 없고,

this.a 를 사용할 수도 없다.

### 요약:

inner class의 객체를 생성할 수 있는 곳은?

outter class의 instance method 에서만...

# static inner class

## static inner class 란?

|  |
| --- |
| class OutterClass {  static class InnerClass {  }  } |

static inner class는, 앞에 static 이 붙은 inner 클래스이다.

static inner class 객체를 생성할 때, outter class 객체에 대한 this 가 전달되지 않는다.

그래서 static inner class의 메소드에서 outter class 객체에 대한 this 를 사용할 수 없다.

outter class 객체에 대한 this 가 없는 곳에서도 (static method),

inner class 객체를 생성할 수 있다.

## 예제

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35 | package net.skhu.inner;  public class OutterClass2 {  int a;  static class InnerClass {  int b;  void method2() {  b = 5;  this.b = 6;  // a = 7; 컴파일 에러  // OutterClass2.this.a = 8; 컴파일 에러  // instanceMethod(); 컴파일 에러  // this.instanceMethod(); 컴파일 에러  // OutterClass2.this.instanceMethod(); 컴파일 에러  main(null);  }  }  public void instanceMethod() {  InnerClass obj = new InnerClass();  a = 1;  this.a = 2;  obj.method2();  }  public static void main(String[] args) {  InnerClass obj = new InnerClass();  obj.method2();  // a = 3; 컴파일 에러  // this.a = 4; 컴파일 에러  }  } |

|  |
| --- |
| // a = 7; 컴파일 에러  // OutterClass2.this.a = 8; 컴파일 에러  // instanceMethod(); 컴파일 에러  // this.instanceMethod(); 컴파일 에러  // OutterClass2.this.instanceMethod(); 컴파일 에러 |

위 코드들이 에러인 이유는,

static inner 클래스에서는 outter 클래스의 this를 사용할 수 없기 때문이다.

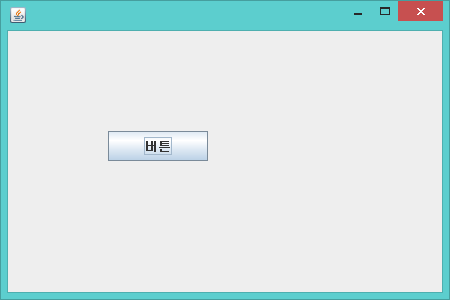
|  |
| --- |
| public static void main(String[] args) {  InnerClass obj = new InnerClass();  obj.method2();  } |

static 메소드인 main 메소드에서도, static inner 클래스 객체를 생성할 수 있다.

# annoymous inner class

## 예제 #1

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37  38  39  40  41  42  43  44  45 | package net.skhu.inner;  import java.awt.event.ActionEvent;  import java.awt.event.ActionListener;  import javax.swing.JButton;  import javax.swing.JFrame;  import javax.swing.JOptionPane;  public class OutterClass3 extends JFrame {  String message = "클릭";  public OutterClass3() {  setBounds(100, 100, 450, 300);  setDefaultCloseOperation(JFrame.EXIT\_ON\_CLOSE);  getContentPane().setLayout(null);  JButton button = new JButton("버튼");  button.setBounds(100, 100, 100, 30);  getContentPane().add(button);  ActionListener listener = new MyActionListener(this);  button.addActionListener(listener);  }  public static void main(String[] args) {  OutterClass3 window = new OutterClass3();  window.setVisible(true);  }  }  class MyActionListener implements ActionListener {  OutterClass3 window;  public MyActionListener(OutterClass3 window) {  this.window = window;  }  @Override  public void actionPerformed(ActionEvent e) {  JOptionPane.showMessageDialog(window, window.message);  }  } |



### 버튼 클릭

버튼이 클릭될 때 실행되는 코드를 구현하려면,

(1) ActionListener 인터페이스를 구현(implements)한 클래스를 만들고,

actionPerformed 메소드를 재정의해야 한다.

|  |
| --- |
| class MyActionListener implements ActionListener {  OutterClass3 window;  public MyActionListener(OutterClass3 window) {  this.window = window;  }  @Override  public void actionPerformed(ActionEvent e) {  JOptionPane.showMessageDialog(window, window.message);  }  } |

(2) 그렇게 구현한 클래스의 객체를 한 개 생성해야 한다.

|  |
| --- |
| ActionListener listener = new MyActionListener(this); |

(3) 생성된 객체를 버튼에 연결해야 한다.

|  |
| --- |
| button.addActionListener(listener); |

이제 버튼이 클릭되면, MyActionListener 클래스의 actionPerformed 메소드가 즉시 호출된다.

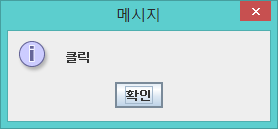
### 메시지 박스

|  |
| --- |
| JOptionPane.showMessageDialog(window, window.message); |

위 코드는 아래와 같은 메시지 박스를 화면에 출력한다.

이 메소드의 첫째 파라미터는, 윈도우 객체이다. 이 윈도우 객체 중앙에 메시지 박스가 위치하게 된다.

이 메소드의 둘째 파라미터는, 화면에 표시할 문자열이다.



## 예제 #2

예제#1 과 예제 #2 는 동일하게 작동하는 코드이다.

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37  38 | package net.skhu.inner;  import java.awt.event.ActionEvent;  import java.awt.event.ActionListener;  import javax.swing.JButton;  import javax.swing.JFrame;  import javax.swing.JOptionPane;  public class OutterClass4 extends JFrame {  String message = "클릭";  public OutterClass4() {  setBounds(100, 100, 450, 300);  setDefaultCloseOperation(JFrame.EXIT\_ON\_CLOSE);  getContentPane().setLayout(null);  JButton button = new JButton("버튼");  button.setBounds(100, 100, 100, 30);  getContentPane().add(button);  ActionListener listener = new MyActionListener();  button.addActionListener(listener);  }  public static void main(String[] args) {  OutterClass4 window = new OutterClass4();  window.setVisible(true);  }  class MyActionListener implements ActionListener {  @Override  public void actionPerformed(ActionEvent e) {  JOptionPane.showMessageDialog(OutterClass4.this, message);  }  }  } |

MyActionListener 클래스를 이너 클래스로 구현하였다.

아우터 클래스의 this가 이너 클래스에 자동으로 전달되기 때문에,

그리고 이너 클래스에서 아우터 클래스의 멤버 변수를 사용할 수 있기 때문에,

MyActionListener 클래스의 코드가 간결해졌다.

## 예제 #3

예제#2 와 예제 #3 은 동일하게 작동하는 코드이다.

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36 | package net.skhu.inner;  import java.awt.event.ActionEvent;  import java.awt.event.ActionListener;  import javax.swing.JButton;  import javax.swing.JFrame;  import javax.swing.JOptionPane;  public class OutterClass5 extends JFrame {  String message = "클릭";  public OutterClass5() {  setBounds(100, 100, 450, 300);  setDefaultCloseOperation(JFrame.EXIT\_ON\_CLOSE);  getContentPane().setLayout(null);  JButton button = new JButton("버튼");  button.setBounds(100, 100, 100, 30);  getContentPane().add(button);  ActionListener listener = new ActionListener() {  @Override  public void actionPerformed(ActionEvent e) {  JOptionPane.showMessageDialog(OutterClass5.this, message);  }  };  button.addActionListener(listener);  }  public static void main(String[] args) {  OutterClass5 window = new OutterClass5();  window.setVisible(true);  }  } |

anonymous inner class 문법은, 코드를 간결하게 구현하기 위해서, 자식 클래스 이름을 생략한다.

anonymous inner class 문법으로 구현하였다.

|  |
| --- |
| ActionListener listener = new ActionListener() {  @Override  public void actionPerformed(ActionEvent e) {  JOptionPane.showMessageDialog(OutterClass5.this, message);  }  }; |

위 코드가 하는 일은 다음과 같다.

(1) ActionListener 인터페이스를 구현(implements)한 자식 클래스를 만들었다.

이 자식 클래스의 이름은 없다. (anonymous class)

(2) 이 자식 클래스에서 actionPerformed 메소드를 재정의하였다.

(3) 이 자식 클래스의 객체가 하나 생성되고,

(4) 그 객체에 대한 참조가 listener 지역 변수에 대입된다.

## 예제 #4

예제 #3 과 예제 #4 는 동일하게 작동하는 코드이다.

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35 | package net.skhu.inner;  import java.awt.event.ActionEvent;  import java.awt.event.ActionListener;  import javax.swing.JButton;  import javax.swing.JFrame;  import javax.swing.JOptionPane;  public class OutterClass6 extends JFrame {  String message = "클릭";  public OutterClass6() {  setBounds(100, 100, 450, 300);  setDefaultCloseOperation(JFrame.EXIT\_ON\_CLOSE);  getContentPane().setLayout(null);  JButton button = new JButton("버튼");  button.setBounds(100, 100, 100, 30);  getContentPane().add(button);  button.addActionListener(new ActionListener() {  @Override  public void actionPerformed(ActionEvent e) {  JOptionPane.showMessageDialog(OutterClass6.this, message);  }  });  }  public static void main(String[] args) {  OutterClass6 window = new OutterClass6();  window.setVisible(true);  }  } |

|  |
| --- |
| button.addActionListener(new ActionListener() {  @Override  public void actionPerformed(ActionEvent e) {  JOptionPane.showMessageDialog(OutterClass6.this, message);  }  }); |

위 코드가 하는 일은 다음과 같다.

(1) ActionListener 인터페이스를 구현(implements)한 자식 클래스를 만들었다.

이 자식 클래스의 이름은 없다. (anonymous class)

(2) 이 자식 클래스에서 actionPerformed 메소드를 재정의하였다.

(3) 이 자식 클래스의 객체가 하나 생성되고,

(4) 그 객체를 파라미터로 전달하면서, button.addActionListener(생성된객체); 메소드가 호출된다.

## 예제 #5

예제 #4 와 예제 #5 는 동일하게 작동하는 코드이다.

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27 | package net.skhu.inner;  import javax.swing.JButton;  import javax.swing.JFrame;  import javax.swing.JOptionPane;  public class OutterClass7 extends JFrame {  String message = "클릭";  public OutterClass7() {  setBounds(100, 100, 450, 300);  setDefaultCloseOperation(JFrame.EXIT\_ON\_CLOSE);  getContentPane().setLayout(null);  JButton button = new JButton("버튼");  button.setBounds(100, 100, 100, 30);  getContentPane().add(button);  button.addActionListener(e -> { JOptionPane.showMessageDialog(OutterClass7.this, message); });  }  public static void main(String[] args) {  OutterClass7 window = new OutterClass7();  window.setVisible(true);  }  } |

람다 익스프레션(lambda expression) 문법을 활용하여 구현하였다.

람다 익스프레션은 Java8에 추가된 문법이다.

|  |
| --- |
| new ActionListener() {  @Override  public void actionPerformed(ActionEvent e) {  JOptionPane.showMessageDialog(OutterClass6.this, message);  }  } |

위 코드와 아래 코드는 완전히 동일하다

|  |
| --- |
| e -> { JOptionPane.showMessageDialog(OutterClass7.this, message); } |

람다 익스프레션 문법은, anonymous inner class 문법에서 좀 더 많은 부분을 생략한다.

메소드 파라미터와 메소드 본문만 남기고, 나머지 부분은 생략한다.

## 예제 #6

예제 #5 와 예제 #6 은 동일하게 작동하는 코드이다.

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27 | package net.skhu.inner;  import javax.swing.JButton;  import javax.swing.JFrame;  import javax.swing.JOptionPane;  public class OutterClass8 extends JFrame {  String message = "클릭";  public OutterClass8() {  setBounds(100, 100, 450, 300);  setDefaultCloseOperation(JFrame.EXIT\_ON\_CLOSE);  getContentPane().setLayout(null);  JButton button = new JButton("버튼");  button.setBounds(100, 100, 100, 30);  getContentPane().add(button);  button.addActionListener(e -> JOptionPane.showMessageDialog(OutterClass8.this, message));  }  public static void main(String[] args) {  OutterClass8 window = new OutterClass8();  window.setVisible(true);  }  } |

|  |
| --- |
| e -> { JOptionPane.showMessageDialog(OutterClass7.this, message); } |

위 코드와 아래 코드는 완전히 동일하다

|  |
| --- |
| e -> JOptionPane.showMessageDialog(OutterClass7.this, message) |

람다 익스프레션에서, 메소드 본문이 간단한 메소드 호출 호출 뿐이라면,

중괄호와 세미콜론을 생략할 수 있다.

람다 익스프레션의 메소드 본문을 표현식(expression) 한 개로 적을 수 있다면,

중괄호와 세미콜론을 생략할 수 있다.