

## Item 9

`try-finally`보다는  
`try-with-resources`를 사용하라

발표자료

# Item 9

## 목차

01 자바의 자원 회수

---

02 try-finally

---

03 try-finally 문제점

---

04 try-with-resources

---

05 try-with-resources 장점

---

06 결론

---

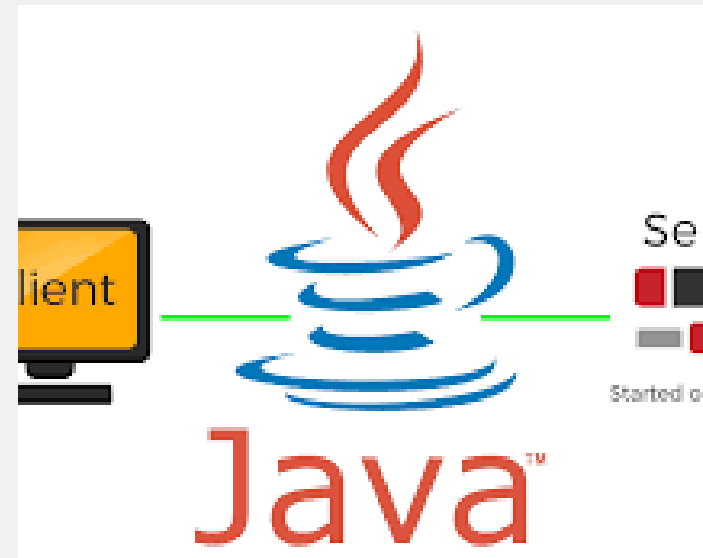
# 자바의 자원 회수

자바 라이브러리에는 close 메서드 호출해 직접 닫아줘야 하는 자원 많다.



## 파일 I/O 관련

FileInputStream,  
FileOutputStream....



## 네트워크 I/O 관련

Socket, URLConnection...



## DB 연결 관련

Connection, Statement...

# 자바의 자원 회수

~~Finalizer~~ —————→ try-finally

```
public class TopLine {  
    static String firstLineOfFile(String path) throws IOException {  
        BufferedReader br = new BufferedReader(new FileReader(path));  
        try {  
            return br.readLine();  
        } finally {  
            br.close();  
        }  
    }  
}
```

# try-finally

예외 처리할 때 사용하는 구문  
보통 try-catch-finally로 많이 사용

## try

예외 발생 가능한 코드 블록 정의  
자원 사용하는 코드 작성

## catch

try 구문에서 예외 발생하면  
실행하는 코드 작성

## finally

try 블록 관계없이 항상  
실행 보장되는 코드 정의  
자원 회수하는 close 메서드 호출

try-finally

~~try-finally~~ —————> ????

why?

# try-finally 문제점

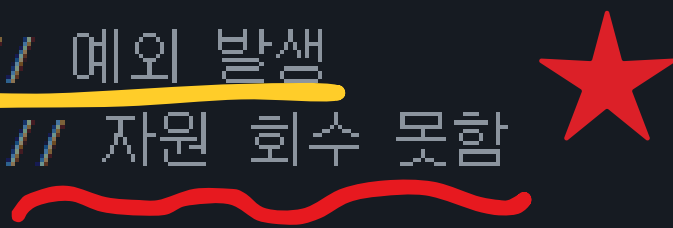
자원 여러 개 사용 시 try-finally  
중첩으로 코드 지저분해짐

```
// 코드 9-2 자원이 둘 이상이면 try-finally 방식은 너무 지저분하다! (47쪽)
static void copy(String src, String dst) throws IOException {
    InputStream in = new FileInputStream(src);
    try {
        OutputStream out = new FileOutputStream(dst);
        try {
            byte[] buf = new byte[BUFFER_SIZE];
            int n;
            while ((n = in.read(buf)) >= 0)
                out.write(buf, 0, n);
        } finally {
            out.close();
        }
    } finally {
        in.close();
    }
}
```

# try-finally 문제점

finally에서 예외 발생 시  
자원 회수 되지 않는 경우 발생

```
static void copy(String src, String dst) throws IOException {  
    InputStream in = new FileInputStream(src);  
    OutputStream out = new FileOutputStream(dst);  
  
    try {  
        byte[] buf = new byte[BUFFER_SIZE];  
        int n;  
        while ((n = in.read(buf)) >= 0)  
            out.write(buf, 0, n);  
  
    } finally {  
        in.close(); // 예외 발생  
        out.close(); // 자원 회수 못함  
    }  
}
```





# try-finally 문제점

예외 여러 개 발생 시  
가장 나중에 발생한 예외만 보임

```
public class Test {  
    public static void main(String[] args) throws Exception {  
        try{  
            install();  
        }catch(Exception e){  
            e.printStackTrace();  
        }  
    }  
  
    public static void install() throws Exception{  
        Bomb bomb = new Bomb();  
        try{  
            bomb.activate();  
            //예외 발생 ✓  
            bomb.process();  
        }finally{  
            //예외 발생 ✓  
            bomb.check();  
            bomb.close();  
        }  
    }  
}
```

추적 결과

```
java.lang.Exception: 폭탄 상태 체크중 예외발생!!!! !@#!@#!@#  
    at Item_9.ex1.Bomb.check(Bomb.java:22)    //check 예외 지점만 출력  
    at Item_9.ex1.Test.install(Test.java:20)  
    at Item_9.ex1.Test.main(Test.java:6)
```

try-with-resources

~~finalizer  
try-finally~~



try-with-resources

`try-with-resources`

**try에 자원 전달하면 try 끝나면 자동으로  
close 메서드 호출해 자원 회수**

# try-with-resources 장점

코드가 짧아지고 읽기 수월해짐

```
// 코드 9-2 자원이 둘 이상이면 try-finally 방식은 너무 지저분하다! (47쪽)
static void copy(String src, String dst) throws IOException {
    InputStream in = new FileInputStream(src);
    try {
        OutputStream out = new FileOutputStream(dst);
        try {
            byte[] buf = new byte[BUFFER_SIZE];
            int n;
            while ((n = in.read(buf)) >= 0)
                out.write(buf, 0, n);
        } finally {
            out.close();
        }
    } finally {
        in.close();
    }
}
```

이전 코드



```
static void copy(String src, String dst) throws IOException {
    try (InputStream in = new FileInputStream(src);
        OutputStream out = new FileOutputStream(dst)) {
        byte[] buf = new byte[BUFFER_SIZE];
        int n;
        while ((n = in.read(buf)) >= 0)
            out.write(buf, 0, n);
    }
}
```

이후 코드

# try-with-resources 장점

## 예외 누락 문제 해결

```
public class Test {  
    public static void main(String[] args) throws Exception {  
        try{  
            install();  
        }catch(Exception e){  
            e.printStackTrace();  
        }  
    }  
  
    public static void install() throws Exception{  
        try ( Bomb bomb = new Bomb());{  
            bomb.activate();  
            //예외발생  
            bomb.process();  
        }  
    }  
}
```

추적 결과

java.lang.Exception: process() 예외발생

at Item\_9.ex1.Bomb.process(Bomb.java:18)

at Item\_9.ex1.Test.install(Test.java:16)

at Item\_9.ex1.Test.main(Test.java:6)

Suppressed: java.lang.Exception: close() 예외발생

at Item\_9.ex1.Bomb.close(Bomb.java:13)

at Item\_9.ex1.Test.install(Test.java:13)

... 1 more

// 집어삼켜진 예외도 출력

# try-with-resources 장점

catch절로 다수 예외 처리 가능

```
// 코드 9-5 try-with-resources를 catch 절과 함께 쓰는 모습 (49쪽)
static String firstLineOfFile(String path, String defaultVal) {
    try (BufferedReader br = new BufferedReader(
        new FileReader(path))) {
        return br.readLine();
    } catch (IOException e) {
        return defaultVal;
    }
}
```

# 결론

자바 자원 회수

~~finalizer  
try-finally~~



try-with-resources

감사합니다

Thank you



# 감사합니다

Thank you