

Item 9

try-finally보다는 try-with-resources를 사용하라

- ✅ 자바 라이브러리에는 `close` 메서드를 호출해 직접 닫아줘야 하는 자원이 많다
- ✅ 자원 닫기는 클라이언트가 놓치기 쉬워서 예측할 수 없는 성능 문제로 이어지기도 한다
- ✅ 전통적으로 자원이 제대로 닫힘을 보장하는 수단으로 `try-finally`가 쓰였다

try-finally 단점

```
package item9;

import java.io.BufferedReader;
import java.io.FileInputStream;
import java.io.IOException;

public class Test {

    public static void main(String args[]) throws IOException {
        FileInputStream is = null;
        BufferedReader bis = null;
        try {
            is = new FileInputStream("file.txt");
            bis = new BufferedReader(is);
            int data = -1;
            while((data = bis.read()) != -1){
                System.out.print((char) data);
            }
        } finally {
            // close resources
            if (is != null) is.close();
            if (bis != null) bis.close();
        }
    }
}
```

해당 방법은 다음과 같은 단점이 존재한다

1. 자원 반납 때문에 코드가 복잡함
2. 실수로 자원을 반납하지 못하는 경우 존재
3. 예외로 자원을 반납하지 못하는 경우 존재
4. 예외 스택 트레이스가 누락되어 디버깅이 어려움

✅ Java는 이러한 문제점을 해결하고자 Java7부터 자원을 자동으로 반납해주는 try-with-resources 문법을 추가하였다

✅ try-with-resources구조를 사용하려면 해당 자원이 AutoCloseable 인터페이스를 구현해야한다

✅ Java 라이브러리와 서드파티 라이브러리들의 수많은 클래스와 인터페이스가 이미 AutoCloseable을 구현하거나 확장해뒀다

try-finally 에러 스택 트레이스 누락 발생

```
package item9;

public class TraceTest implements AutoCloseable{ no usages new * Complexity is 3 Everything is cool!

    @Override new *
    public void close() throws RuntimeException{
        System.out.println("닫힘");
        throw new IllegalStateException();
    }

    public void traceTest() { no usages new *
        System.out.println("트레이스 테스트");
        throw new IllegalStateException();
    }
}
```

✓ TraceTest클래스의 close() ,traceTest()

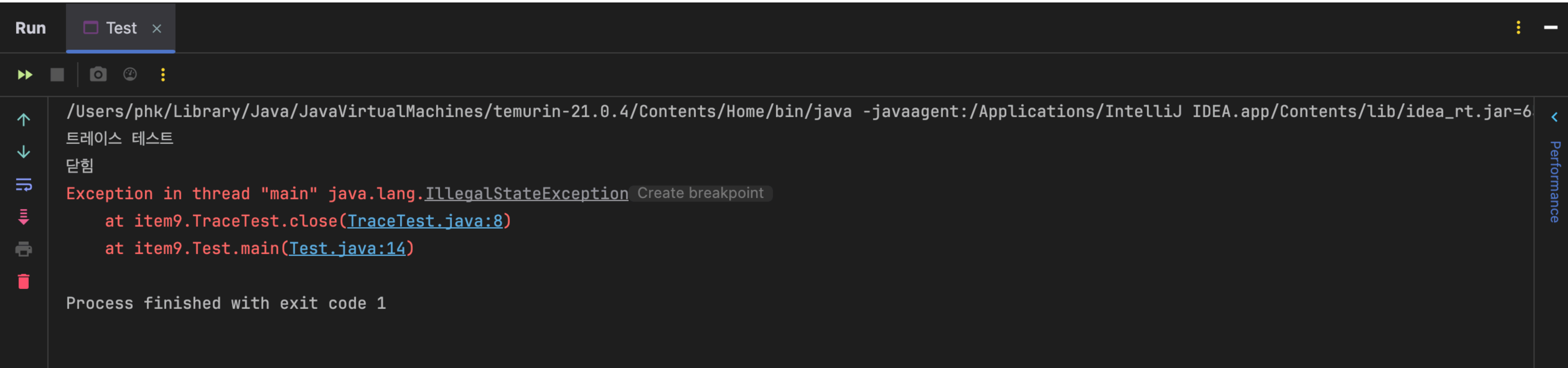
메서드는 IllegalStateException 던진다

```
1 package item9;
2
3 import java.io.IOException;
4
5 ▶ public class Test { new * Complexity is 4 Everything is cool!
6
7 ▶     public static void main(String args[]) throws IOException { new * Complexity is 3 Everything is cool!
8         TraceTest test = null;
9         try {
10             test = new TraceTest();
11             test.traceTest(); // 트레이스 누락
12         } finally {
13             if (test != null) {
14                 test.close();
15             }
16         }
17     }
18 }
19
```

✓ main() 메서드에서 close() ,traceTest()

호출한다

try-finally 에러 스택 트레이스 누락 발생



The screenshot shows the Run console in IntelliJ IDEA. The top bar indicates the 'Run' configuration is selected. The console output shows the command to run the application, followed by the execution of '트레이스 테스트' (Trace Test). An exception is thrown: 'Exception in thread "main" java.lang.IllegalStateException'. The stack trace shows the exception was thrown at 'item9.TraceTest.close(TraceTest.java:8)' and 'item9.Test.main(Test.java:14)'. The process finished with exit code 1.

```
/Users/phk/Library/Java/JavaVirtualMachines/temurin-21.0.4/Contents/Home/bin/java -javaagent:/Applications/IntelliJ IDEA.app/Contents/lib/idea_rt.jar=6  
트레이스 테스트  
달함  
Exception in thread "main" java.lang.IllegalStateException Create breakpoint  
    at item9.TraceTest.close(TraceTest.java:8)  
    at item9.Test.main(Test.java:14)  
  
Process finished with exit code 1
```

에러 트레이스가 close() 호출 시에 발생한 예외만 찍힌것을 확인 할 수 있다.

이러한 문제가 발생한다면, 디버깅하는데 상당히 많은 시간이 소요될 것이다.

하지만 try-with-resources 구문은 누락되는 에러 트레이스 없이 모두 남길 수 있다.

try-with-resources 에러 스택 트레이스 누락 방지

```
1 package item9;
2
3 import java.io.IOException;
4
5 public class Test { new *
6
7     public static void main(String args[]) throws IOException { new *
8         try (TraceTest test = new TraceTest()) {
9             test.traceTest(); // 트레이스 누락 X
10        }
11    }
12 }
13
```

코드도 더욱 간결해지고 close(), traceTest()의 에러 트레이스 모두 찍힌것을 확인할 수 있다

```
Run Test x
>> [Icons]
/Users/phk/Library/Java/JavaVirtualMachines/temurin-21.0.4/Contents/Home/bin/java -javaagent:/Applications/IntelliJ IDEA.app/Contents/lib/idea_rt.jar=6
트레이스 테스트
달함
Exception in thread "main" java.lang.IllegalStateException Create breakpoint
    at item9.TraceTest.traceTest(TraceTest.java:13)
    at item9.Test.main(Test.java:9)
Suppressed: java.lang.IllegalStateException
    at item9.TraceTest.close(TraceTest.java:8)
    at item9.Test.main(Test.java:8)

Process finished with exit code 1
Performance
```

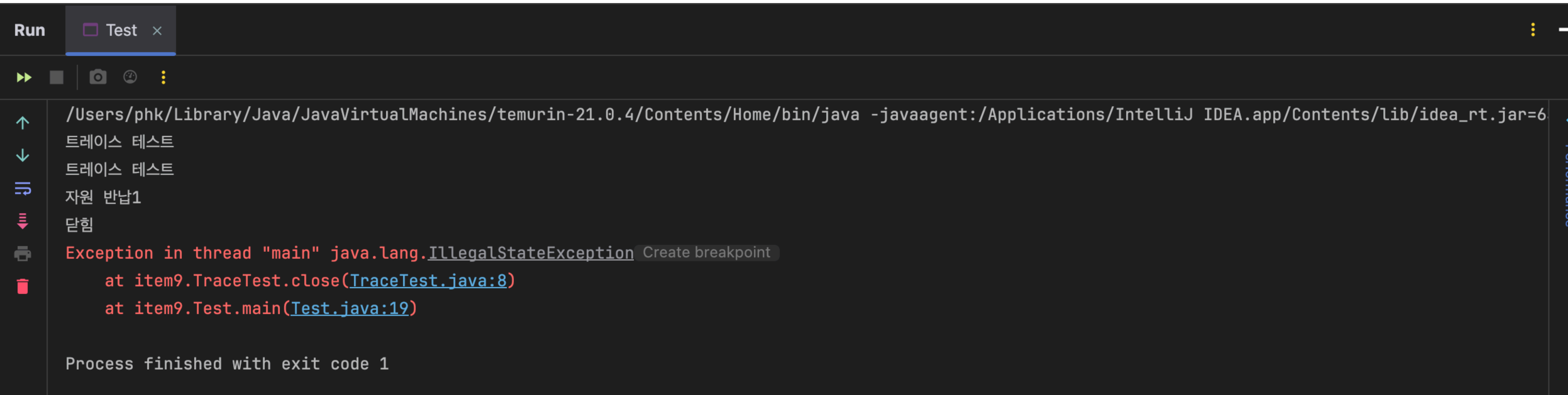
try-finally 에러로 자원을 반납하지 못하는 경우

```
1 package item9;
2
3 public class TraceTest implements AutoCloseable{ 2 usages new * Complexity is 3 Everything is cool!
4
5     @Override new *
6     public void close() throws RuntimeException{
7         System.out.println("닫힘");
8         throw new IllegalStateException();
9     }
10
11     public void traceTest() { 1 usage new *
12         System.out.println("트레이스 테스트");
13     }
14 }
15
```

```
5 public class Test { new * Complexity is 6 It's time to do something...
6
7     public static void main(String args[]) throws IOException { new * Complexity is 5 Everything is cool!
8         TraceTest test1 = null;
9         TraceTest test2 = null;
10
11         try {
12             test1 = new TraceTest();
13             test2 = new TraceTest();
14             test1.traceTest();
15             test2.traceTest();
16         } finally {
17             if (test1 != null) {
18                 System.out.println("자원 반납1");
19                 test1.close();
20             }
21
22             if (test2 != null) {
23                 System.out.println("자원 반납2");
24                 test2.close();
25             }
26         }
27     }
28 }
```

자원을 사용하고 finally 구문에서 null이 아닌 경우
를 검사하여 자원을 반납한다

try-finally 에러로 자원을 반납하지 못하는 경우



```
Run Test ×
/Users/phk/Library/Java/JavaVirtualMachines/temurin-21.0.4/Contents/Home/bin/java -javaagent:/Applications/IntelliJ IDEA.app/Contents/lib/idea_rt.jar=6
트레이스 테스트
트레이스 테스트
자원 반납1
달힘
Exception in thread "main" java.lang.IllegalStateException Create breakpoint
    at item9.TraceTest.close(TraceTest.java:8)
    at item9.Test.main(Test.java:19)
Process finished with exit code 1
```

두 번째 자원인 test2가 정상적으로 반납되지 않는다.

그 이유는 finally 구문의 test1.close()가 IllegalStateException를 던지는데, 이를 catch하는 코드가 없기 때문이다.

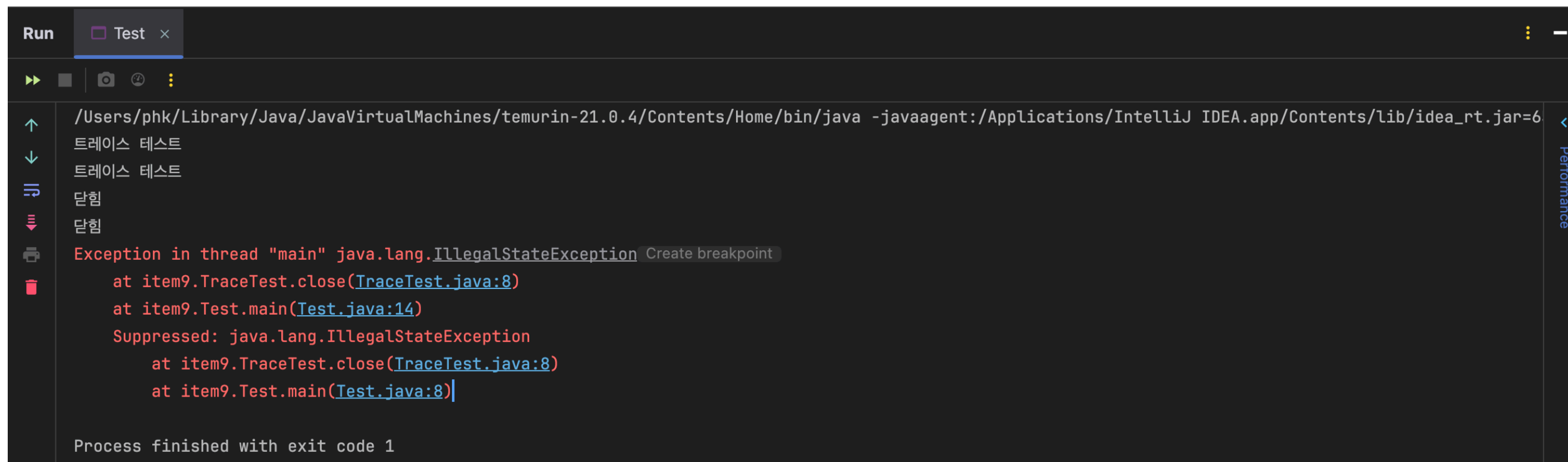
따라서 이를 방지하기 위해서는 test1.close(), test2.close()를 각각 try-catch-finally로 묶어야 하는데, 이는 코드를 더욱 복잡하게 만든다.

try-with-resources 모든 자원 반납

```
3 import java.io.IOException;
4
5 public class Test { new *
6
7     public static void main(String args[]) throws IOException { new *
8     try (
9         TraceTest test1 = new TraceTest();
10        TraceTest test2 = new TraceTest();
11    ) {
12        test1.traceTest();
13        test2.traceTest();
14    }
15 }
16 }
17 }
```

try-with-resources를 사용하면 2가지 자원 모두 정상적으로 반납된다.

그 이유는 Java 파일이 Class 파일로 컴파일 될 때 try-with-resources에서 모든 경우를 try-catch-finally로 변환해주기 때문이다.



try-with-resources 모든 자원 반납

```
3 import java.io.IOException;
4
5 public class Test { new *
6
7     public static void main(String args[]) throws IOException { new *
8         try (
9             TraceTest test1 = new TraceTest();
10            TraceTest test2 = new TraceTest();
11        ) {
12            test1.traceTest();
13            test2.traceTest();
14        }
15    }
16 }
17
```

```
// class version 65.0 (65)
// access flags 0x21
public class item9/Test {

    // compiled from: Test.java

    // access flags 0x1
    public <init>()V
        L0
        LINENUMBER 5 L0
        ALOAD 0
        INVOKESPECIAL java/lang/Object.<init> ()V
        RETURN
    L1
        LOCALVARIABLE this Litem9/Test; L0 L1 0
        MAXSTACK = 1
        MAXLOCALS = 1

    // access flags 0x0
    public static main([Ljava/lang/String;)V throws java/io/IOException
        TRYCATCHBLOCK L0 L1 L2 java/lang/Throwable
        TRYCATCHBLOCK L3 L4 L5 java/lang/Throwable
        TRYCATCHBLOCK L6 L7 L8 java/lang/Throwable
        TRYCATCHBLOCK L9 L10 L11 java/lang/Throwable
        L12
        LINENUMBER 9 L12
        NEW java/io/TraceTest
        ALOAD 0
        INVOKESPECIAL java/io/TraceTest.<init> ()V
        ALOAD 0
        INVOKESPECIAL java/io/TraceTest.traceTest ()V
        ALOAD 0
        INVOKESPECIAL java/io/TraceTest.close ()V
        ALOAD 0
        INVOKESPECIAL java/io/TraceTest.traceTest ()V
        ALOAD 0
        INVOKESPECIAL java/io/TraceTest.close ()V
        RETURN
    L13
        LINENUMBER 13 L13
        RETURN
}
```

Bytecode를 보면 컴파일러가 모든 메서드의 예외의 경우를 try-catch-finally(TRYCATCHBLOCK)으로 변환해주는것을 확인할 수 있다.

TRYCATCHBLOCK은 test1.traceTest(), test1.close(), test2.traceTest(), test2.close()의 대한것으로 총 4개가 생성된다.

try-with-resources 를 사용해야하는 이유

- ✓ 코드를 간결하게 만들 수 있음**
- ✓ 모든 에러 스택 트레이스를 남겨서 디버깅을 수월하게 할 수 있음**
- ✓ 에러 및 실수로 자원을 반납하지 못하는 경우를 방지할 수 있음**