아이템09 (try-finally보다는 try-with-resources를 사용하라)

자바 라이브러리에는 close 메서드를 호출해 직접 닫아줘야 하는 자원이 많다. ex) InputStream, OutputStream. 하지만 자원 닫기는 클라이언트가 놓치기 쉬워서 예측할 수 없는 성능 문제로 이어지기도 한다. 안전망으로 finalizer 를 활용하고는 있지만 finalizer 가 믿음직하지 못하다.

이를 위한 해결책으로 전통적으로 자원의 닫힘을 보장하는 수단인 try-finally 가 쓰였다.

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

닫아야 하는 자원이 하나뿐인 try-finally를 사용한 close

하지만 자원이 많아지면 다음과 같은 문제가 발생한다.

```
static void lineOfFile(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, off: 0, n);
        } finally {
            out.close();
        }
    } finally {
        in.close();
    }
}
```

닫아야 하는 자원이 여러 개인 try-finally를 사용한 close

단점

1. 자원이 많아지면 코드가 복잡해진다.

2. try부분에서 발생한 예외가 finally부분에서 발생한 예외를 삼켜 디버깅을 어렵게 만든다. stackTrace에도 이에 관한 정보가 남지 않는다.

위와 같은 문제를 해결하기 위해 try-with-resources를 사용하자. 이 구조를 사용하려면 해당 자원이 AutoCloseable 인터페이스를 구현해야 한다. 닫아야 하는 자원 클래스를 작성할 땐 AutoCloseable을 반드시 구현하는 것이 좋다.

```
static void lineOfFile(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, off: 0, n);
    }
}
```

try-with-resources를 사용한 close

try-with-resources 버전이 짧고 읽기 수월하며 문제를 진단하기도 좋다. 위의 write와 close(코드에는 나타나지 않는다) 양쪽에서 예외가 발생하면 close에서 발생한 예외는 숨겨지고 write에서 발생한 예외가 기록되는데 이는 스택 추적내역에 "suppressed" 꼬리표를 달고 출력된다. 참고로 try-with-resources에서도 catch절 사용이 가능하다.

정리

꼭 회수해야 하는 자원을 다룰 때 try-finally 말고 예외 없이 try-with-resources를 사용하자. 코드가 짧고 분명해지며 예외 정보 또한 훨씬 유용해진다. try-finally로 작성하면 코드가 지저분해지지만 try-with-resources로는 정확하고 쉽게 자원 회수가 가능하다.