[2기][item9] try-finally보다는 try-with-resources를 사용하라

close 메서드를 호출해 직접 자원을 닫아줘야 하는 클래스

- InputStream
- OutputStream
- java.sql.Connection
- 자원을 닫아주지 않으면 예측할 수 없는 성능 문제로 이어지기도 한다

자원을 닫아주지 않으면 어떤 문제가 생길 수 있는가

iops 1k
2017-06-29 10:23:28

아주 간단히만 설명하면

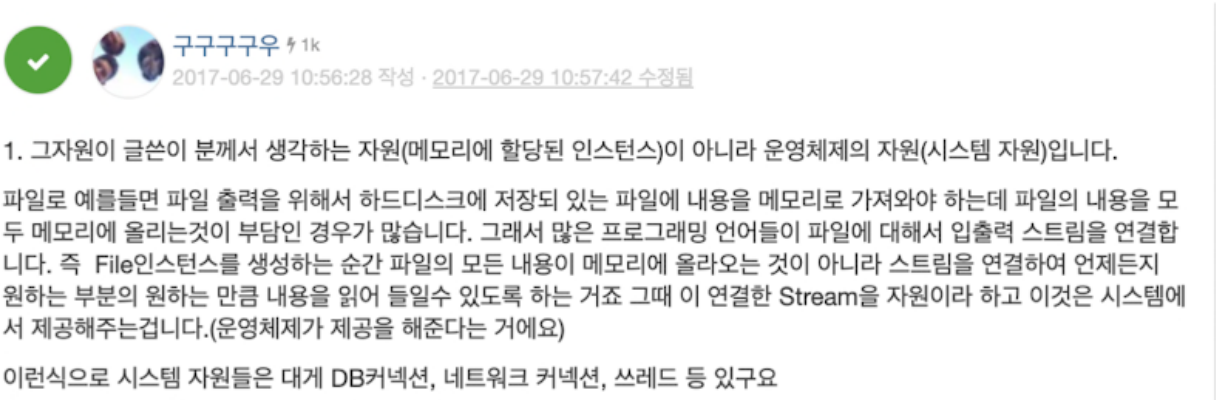
우리가 프로그래밍 하고 있는것들은 memory를 제어하고 있는것입니다.

우리가 작업하고 있는 변수나 instance들은 memory에 있는 value와 address를 제어하는 것인데

File 은 hdd에 존재하는것 이므로 외부에 있는 자원이고 이 자원을 쓰려면 외부 자원을 open을 해서

메모리로 가지고 와야하며 다 사용하고 나면 다시 연결을 해제(close)해줘야 합니다.

마찬가지로 Connection도 외부 자원이고 쓰고나면 close를 해줘야합니다.



구구구구우 1k
2017-06-29 10:56:28 작성 · 2017-06-29 10:57:42 수정됨

1. 그자원이 글쓴이 분께서 생각하는 자원(메모리에 할당된 인스턴스)이 아니라 운영체제의 자원(시스템 자원)입니다.

파일로 예를들면 파일 출력을 위해서 하드디스크에 저장되 있는 파일에 내용을 메모리로 가져와야 하는데 파일의 내용을 모두 메모리에 올리는것이 부담인 경우가 많습니다. 그래서 많은 프로그래밍 언어들이 파일에 대해서 입출력 스트림을 연결합니다. 즉 File인스턴스를 생성하는 순간 파일의 모든 내용이 메모리에 올라오는 것이 아니라 스트림을 연결하여 언제든지 원하는 부분의 원하는 만큼 내용을 읽어 들일수 있도록 하는 거죠 그때 이 연결한 Stream을 자원이라 하고 이것은 시스템에서 제공해주는겁니다.(운영체제가 제공을 해준다는 거에요)

이런식으로 시스템 자원들은 대게 DB커넥션, 네트워크 커넥션, 쓰레드 등 있구요

https://okky.kr/article/401102

전통적으로 자원을 닫아주는 방법

- try 블록 이전에 자원을 생성 후 finally 문에서 close() 를 호출한다
- 자원이 둘 이상이면 try-finally 방식은 너무 지저분하다

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

```
// try-finally is ugly when used with more than one resource! (Page 34)
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();
    }
}
```

게다가 단순해보이는 이 코드는 결점이 있다

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

▼ 뭘까요?

```
static String firstLineOfFile(String path) throws IOException {
    BufferedReader br = new BufferedReader(new FileReader(path));
    try {
        return br.readLine(); // (1)
    } finally {
        br.close(); // (2)
    }
}
```

- 예외는 (1)과 (2)에서 모두 발생할 수 있는데, 만약 두 곳에서 (1), (2) 차례로 예외가 나는 경우 (2)의 예외가 (1)의 예외를 완전히 집어 삼켜 버려 스택 추적 내역에서 (1)의 정보는 남지 않는다
- 기기에 물리적인 문제가 생긴다면 (1), (2) 에서 모두 예외가 발생한다
- 이는 디버깅을 아주 어렵게 한다는 문제점이 있다!
- 물론 (1)의 예외를 대신 기록하도록 코드를 수정할 순 있지만 코드가 너무 지저분해진다

```
public class NetworkConnection {

    public void doSomething() throws Exception {
        throw new Exception("throw exception while doing something");
    }

    public void close() throws IOException {
        throw new Exception("throw exception while closing");
    }

}
```

```
public class NetworkConnectionTest {

    static void test() throws Exception {
        NetworkConnection networkConnection = new NetworkConnection();
        try {
            networkConnection.doSomething();
        } finally {
            networkConnection.close();
        }
    }

    public static void main(String[] args) throws IOException {
        test();
```

```
    }

}
```

```
Exception in thread "main" java.lang.Exception Create breakpoint : throw exception while closing
    at effectivejava.chapter2.item9.tryfinally.NetworkConnection.close(NetworkConnection.java:10)
    at effectivejava.chapter2.item9.tryfinally.NetworkConnectionTest.test(NetworkConnectionTest.java:12)
    at effectivejava.chapter2.item9.tryfinally.NetworkConnectionTest.main(NetworkConnectionTest.java:22)
```

보통 처음 발생한 예외인 throw exception while doing something 예외 메세지를 보고 싶을 텐데 close 에서 발생할 예외 메세지만 출력된다

Java7. try-with-resources

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

```
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);
    }
}
```

- 코드가 더 짧아져서 가독성이 좋다

▼ 단, try 문에서 사용되는 자원이 AutoCloseable 인터페이스를 구현해야 한다

```
public class BufferedReader extends Reader

public abstract class Reader implements Readable, Closeable

public interface Closeable extends AutoCloseable
```

```
package java.lang;

public interface AutoCloseable {
```

```
    void close() throws Exception;
}
```

▼ 문제를 진단하기도 좋다

```
public class NetworkConnection implements AutoCloseable {

    public void doSomething() throws Exception {
        throw new Exception("throw exception while doing something");
    }

    public void close() throws Exception {
        throw new Exception("throw exception while closing");
    }

}
```

```
public class NetworkConnectionTest {

    static void test() throws Exception {
        try(NetworkConnection networkConnection = new NetworkConnection()) {
            networkConnection.doSomething();
        }
    }

    public static void main(String[] args) throws Exception {
        test();
    }

}
```

- `doSomething()` 과 `close()` 호출시 모두 예외가 발생하면 아래와 같이 콘솔에 찍힌다.

```
Exception in thread "main" java.lang.Exception Create breakpoint : throw exception while doing something
    at effectivejava.chapter2.item9.tryfinally.NetworkConnection.doSomething(NetworkConnection.java:6)
    at effectivejava.chapter2.item9.tryfinally.NetworkConnectionTest.test(NetworkConnectionTest.java:15)
    at effectivejava.chapter2.item9.tryfinally.NetworkConnectionTest.main(NetworkConnectionTest.java:20)
    Suppressed: java.lang.Exception: throw exception while closing
        at effectivejava.chapter2.item9.tryfinally.NetworkConnection.close(NetworkConnection.java:10)
        at effectivejava.chapter2.item9.tryfinally.NetworkConnectionTest.test(NetworkConnectionTest.java:14)
        ... 1 more
```

- `close()` 시 발생한 예외는 숨겨지고 `doSomething()` 에서 발생한 에외가 기록된다
- 숨겨진 예외도 스택 추적 내역에 Suppressed 라는 꼬리표를 달고 출력된다.

- 또한 보통의 try-finally 문처럼 try-with-resources 문도 catch 를 사용할 수 있다.