예외 처리로 죽지않는 프로그램을!

1111 손정우

예외 상황

- 0으로 나누기
- 메모리 부족
- 잘 못된 입력
- 존재하지 않는 파일
- 서버와의 연결 실패

…

조건문으로 처리

```
double div(int x, int y) { //x를 y로 나눈다
    return (double)x/(double)y;
}
```

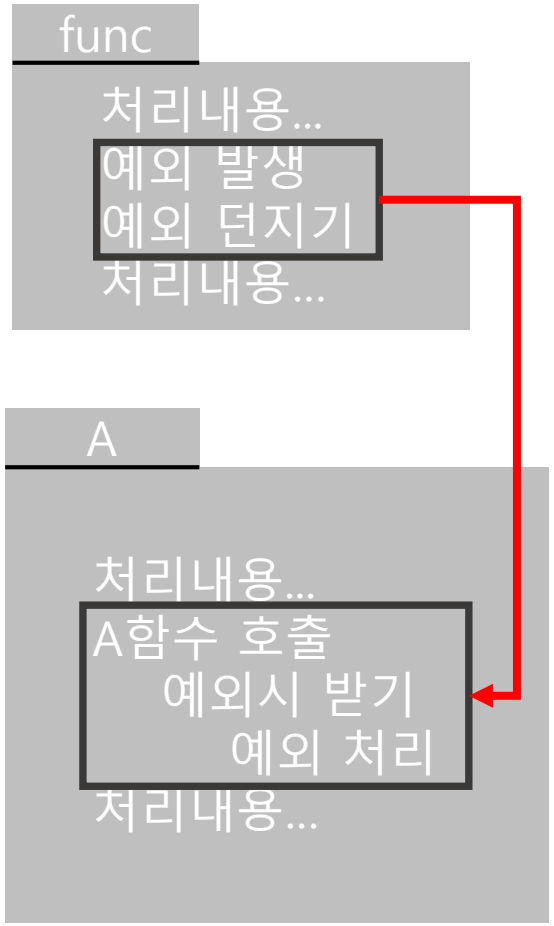
```
double div(int x, int y) { //x를 y로 나눈다
    if(y == 0) return 0;
    return (double)x/(double)y;
}
```

오류가 발생했는지 알 수가 없다.

```
int div(int x, int y, double* res) { //x를 y로 나눈다
    if(y == 0) return 0;
    (*res) = (double)x/(double)y;
    return 1;
}//리턴값이 1이면 성공, 0이면 실패
```

리턴값이 함수의 직관적인 결과값이 아니다.

예외 처리 과정



프로그램을 실행하다가 예외가 발생한다

예외가 발생한 함수(func)에서 호출한 쪽(A)으로 예외를 던진다

함수를 호출한 곳(A)에서 예외를 받는다

함수를 호출한 곳(A)에서 예외를 처리한다

예외 던지기

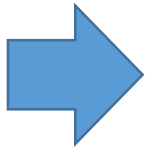
```
int div(int x, int y) { //x를 y로 나눈다
    if(y == 0)
        throw 0;
    return (double)x/(double)y;
}
```

throw (던질 예외의 내용); 으로 예외를 던질 수 있다.

예외 객체를 만들어 던지기

```
class DivideByZeroException{
};
```

```
int div(int x, int y) { //x를 y로 나눈다
    if(y == 0)
        throw new DivideByZeroException;
    return (double)x/(double)y;
}
```



예외에 대한 객체를 따로 만들어 처리
-> 예외에 대한 코드 상의 의미을 가지게 함
-> 예외에 대한 다양한 정보를 담을 수 있음

예외 받기

```
class DivideByZeroException{
    print() { ... } //예외에 대한 내용을 출력
};
```

```
int div(int x, int y) { //x를 y로 나눈다
    if(y == 0)
        throw new DivideByZeroException;
    return (double)x/(double)y;
}
```

```
int main() {
    try{
        double res = div(5, 0);
        std::cout<< "결과 : " << res << std::endl;
    } catch (DivideByZeroException& e){
        e.print();
        std::cout << "0으로 나눌 수 없습니다." << std::endl;
    }
    return 0;
}
```

try 블록 안에 있는 작업을 시도한다

예외가 발생하면 작업을 중단

예외 타입과 같은 타입의 매개변수를 가지는
catch로 이동
catch(...) {} 일 경우 모든 종류의 예외를 처리

catch에서 예외를 처리함
(현재 위치에서 해결 할 수 없을 시 다시 throw 가능)

장점

- 여러가지 예외에 대해 대응한다.
- 당장 해결 불가능한 예외를 나중에 해결하도록 미룰 수 있다.
- 주요 로직과 예외 처리 로직을 분리할 수 있다.

단점

- 무분별한 사용으로 프로그램이 올바르게 동작하지 않을 수 있다.
- 생성된 예외를 제대로 처리하지 않으면 한참 후에 여러 예외가 쌓여 심각한 문제를 야기한다.