Exception

- try~catch구문
- BizException
- try~catch~finally 구문
- try~with~resource 구문

try{

에러가 발생할 가능성이 있는 코드

} catch(

에러 발생 시 실행되는 코드

에러 메세지, 에러가 발생한 위치 등을 확인하기 위한 코드 작성

)

```
int[] intArray = {1,2,3};
// intArray 에 1,2,3을 담는 배열을 생성한 후
// 인덱스를 벗어나는 오류를 일부러 일으킴
try{
      System.out.println(intArray[10]);
      System.out.println("인덱싱이 끝났습니다.");
  }catch (ArrayIndexOutOfBoundsException aioobe){
      // 해당 예외 객체에 대한 정보를 볼 수 있음
      // (에러 메시지, 에러가 발생한 위치)
      aioobe.printStackTrace();

  }
```

```
  // try~catch로 예외처리하면 에러가 발생해도 프로그램이 중단되지 않는다.
  System.out.println("인덱싱 예외처리 성공적~!");
```

```
try{
      System.out.println(intArray[10]);
  } catch (Exception exception){
      // Exception은 모든 예외 클래스의 최상위 객체이므로
        // 어떤 예외가 발생하든 catch 된다.
      exception.printStackTrace();

      // 에러가(빨간글씨) 콘솔창에 출력되는 시점과
      // println()이 콘솔창에 출력되는 시점이 다르다.
      System.out.println("빨간글씨 위");
      System.err.println("딸기맛");
      System.out.println("빨간글씨 아래");
```

ArrayIndexOutOfBoundsException 오류가 발생

→ catch 에서 오류를 파악한 후

→ aioobe.printStackTrace(); 코드로 인해 에러가 발생한 위치와 에러 메세지 출력

→ try~catch 구문이 끝난 후 System.out.println("인덱싱 예외처리 성공적~!"); 실행

개발 후반부에 예외 처리하는 것을 권장

```
Scanner scan = new Scanner(System.in);

 while (true){
     System.out.println("행동선택");
     System.out.println("1. 밥먹기 | 2. 잠자기 | 3. 종료");
     System.out.println(">>> ");

    int command = 0;
    try {
        command = Integer.parseInt(scan.nextLine());
      } catch (NumberFormatException e) {
         System.out.println("숫자만 입력해!!");
```

```
            continue;
        }

        if (command == 1){
            System.out.println("냠냠");
        } else if (command == 2) {
            System.out.println("쿨쿨");
        } else if (command == 3){
            break;
    }else {
            System.out.println("숫자는 1~3 중에 하나만 입력해");
        }
    }
```

catch는 여러 개 사용할 수 있다

```
try{
        System.out.println(intArray[100]);
    }catch (ArrayIndexOutOfBoundsException e){
        // 예측 가능한 예외 처리에 대한 catch
        System.out.println("배열의 인덱스를 넘어감");
    }catch (Exception e){
        // 예측하지 못한 예외 상황 발생 시 처리를 위한 catch (최후의 보
        System.out.println("예측하지 못한 예외발생");
    }
```

BizException

BizException

일반적으로 예측 가능한 예외 상황 처리를 위해
따로 Exception을 상속받은 예외 클래스를 만들어서 사용

try~catch~finally 구문

```
try {
            ExMethod.printName("123");
        } catch (BizException e) {
            e.printStackTrace();
        } finally {
            System.out.println("에러가 나든 말든 실행");
        } System.out.println("탈출");

// e.printStackTrace() 와 finally부분, 마지막 "탈출"까지 출력된다.
// finally는 주로 자원을 닫기 위한 목적으로 사용된다.
// .close()
```

try~with~resource 구문

- 자원정리 .close( )가 권장되는 객체를 try() 소괄호 안에 선언 후,

  try( ){ } 중괄호 안에서 사용
- 이후 별도로 .close( ) 코드를 작성하지 않아도 알아서 닫는다.
- 일반적인 try~catch 구문과 달리 catch 부분이 없어도 된다.

```
try(Scanner scan = new Scanner(System.in)){
            int command = Integer.parseInt(scan.nextLine());
        } catch (Exception e){
            System.out.println("숫자가 아닌 걸 입력함");
        }

Scanner copy = new Scanner(System.in); // try~with~resource 구
                                       copy.nextLine(); // 객체 입
```

퀴즈

try~catch~finally 의 실행 순서를 맞추는 문제

```
try {
            qMethod(-1);
            System.out.println("B");
        } catch (Exception e) {
            System.out.println("C");
        }finally {
            System.out.println("D");
        }
        System.out.println("E");
    }

   public static void qMethod(int no) throws Exception {
        if(no < 0){
            throw new Exception();
        }
        System.out.println("A");
// Exception으로 던진 후 println은 실행되지 않음!
```

정답 : C→D→E