예외처리(Exception) try~catch문
error
1. 컴파일 에러: 실행되지 않음
2. 런타임 에러: 실행 중에 오류가 남 *** 위험
-> 치명적인 오류 -> error(프로그램을 중단)
-> 무시할 수 있는 정도의 오류 -> exception(예외)

예외처리(Exception) : 오류가 있어도 중단 되지 않고 프로그램을 수행 *** 예외처리의 목적
1. "예외를 처리하고 정상적으로 프로그램을 수행할 목적" *****
2. try~catch문 사용 -> 예외처리문
3. 데이터베이스 연결 시 등등에 사용한다.

에러
1. 컴파일 에러 -> 프로그램이 실행되지 않음
2. 실행 중 에러(런타임 에러)
1) error(에러) -> 프로그램 중단
2) Exception(예외) -> 예외처리 등록된 에러들 중에...

Exception(예외) -> 예외처리 -> 예상 하고 대비
- 실행 중 오동작이나 결과에 악영향을 미치는 예상치 못한 상황이 발생
- 자바에서는 실행 중 발생하는 에러를 예외로 처리한다(미리 예상, 대비)
  -> error(치명적인에러)가 아닌 무시할수 있는 에러(Exception,예외)
- try문에서 예외가 발생되면 그 아래 명령문이 실행되지 않는다
*** "예외가 생길 것을 대비하여 미리 예외처리문을 만들고 예외가 발생 되지 않으면
  정상적으로 실행되고 예외가 발생이 되면 예외를 처리하고 정상 실행"
*** 예외에 맞게 처리문 작성하지 않으면 프로그램 정상종료 할수 없다.

실행 중 예외가 발생하면
- 자바 플랫폼은 응용프로그램이 예외를 처리하도록 호출
- 응용프로그램이 예외를 처리하지 않으면 프로그램을 강제 종료 시킴

예외발생 경우(예시)
- 정수를 0으로 나누는 경우
- 배열의 크기보다 큰 인덱스로 배열의 원소를 접근하는 경우
- 정수를 읽는 코드가 실행되고 있을 때 사용자가 문자를 입력한 경우
- 클래스를 찾지 못 할 경우(JDBC 드라이버) ****
- 데이터베이스 연동시 커넥션을 오류인경우 ****
** 예외처리 방법 -> 정상 실행,종료
- 발생한 예외에 대해 개발자가 작성한 프로그램 코드에서 대응하는 것

1. 예외 처리 방법
***　　try-catch-finally문　사용
-> finally 블록은 생략 가능
** 예외가 발생 안했을 때 -> 1) -> 3)->4)
** 예외가 발생 하면 -> 1) ->2)"예외를처리" -> 3)-> 4)

```
try{
        1)// 예외가 발생할 가능성이 있는 실행문(명령문) 예외1
} catch(처리할 예외 타입 선언) {
        2)//예외처리문　 예외1
        // 예외가 발생 할 때만 실행
} finally {
        3)//예외 발생 여부와 상관없이 무조건 실행되는 문장
        // 필수　DB연동, 파일처리
}
        4)　 //정상 실행
```

*** catch　{}
printStackTrace() : 예외 발생 시 호출스택에 있었던 메서드의 정보와 예외메세지를 출력
getMessage()　　: 발생한 예외 클래스의 인스턴스에 저장된 메시지(실제 예외)

**다중예외 처리

*** Exception e -> 모든 예외를 처리

```
try {
        // 예외 발생 예상

} catch(Exception e ){
        e.printStakcTrace();
}finally{
            Systen.out.ptintln("기본 실행")
}
```

==========try~catch문을 쓸 때는 기본적으로 이렇게 쓴다
　　발생할만한 오류를 모두 기술하고 마지막에 Exception

```
try {

} catch(ArithmeticException e) {
```

```
        e.printStackTrace();
} catch(InputMismatchException e) {
        e.prinsStackTrace();
} catch(ArrayIndexOutOfBoundsException e) {
        e.prinsStackTrace();
} catch(Exception e) {
        e.printStakcTrace();
}finally{
            Systen.out.ptintln("기본 실행")
}
```

==========

ClassNotFoundException -> class 찾기
SQLException -> SQL관련, Connection

==========

2. 예외 처리 방법
** 예외를 발생 -> 예외를 던졌다, throw, throws
throw new Exception();

```
1. try {
        //고의 예외발생 -> DB 관련 처리시 특별한 경우
        throw new Exception();
} catch(Exception e) {

}
```

** 매서드에 예외를 던졌다. throws 예외1, 예외2,,,,
-> 매서드 선언부에서 예외를 던지면 구현부에서 그 예외가 발생이 되면 try~catch문을 사용하지 않아도 된다.

```
public void excuteQueryService()
        throws ClassNotFoundException,SQLException{
        //try~catch문 생략 가능

}
```