Java Fundamental

# 예외 처리

# 예외

- 프로그램 동작 중 발생하는 **에러**, 혹은 **오류**를 일컫는 말
- 오류가 발생하면 **예외 객체**(exception object)를 생성
- 예외 객체를 이용해 **예외 처리** 가능

# 주요 예외

- **java.lang.Throwable**을 상속받는 예외 클래스

| 종류 | 설명 |
| --- | --- |
| ArithmeticException | 산술 연산 관련 예외(0 나누기 등) |
| ClassCastException | 객체의 형 변환 관련 예외 |
| IllegalArgumentException | 전달인자 관련 예외(parseInt() 등) |
| IndexOutOfBoundsException | 배열 등의 범위를 벗어난 참조 시 |
| NullPointerException | null 참조 시 발생하는 예외 |

# try ~ catch

- **예외를 처리하는 블록**
- **try** 블록을 통해 코드의 실행을 **시도**
- 예외가 발생하면 **catch** 블록을 통해 **처리**
- 어떤 예외 클래스를 이용해서 처리할 것인지 명시

# try ~ catch

```
try {
    // 시도할 코드
} catch ([처리할 예외 클래스명] [변수명]) {
    // 처리할 내용
}
```

# finally

- try ~ catch 블록 실행 후 **반드시 실행되는 블록**
- 주로 **자원의 반납**을 다룸
  - **자원**: 스트림, 파일 입출력 등에서 사용하는 일부 메모리
- finally 블록에서 try ~ catch 블록 작성 가능

# finally

```
try {
    // 시도할 코드
} catch ([예외 클래스명] [변수명]) {
    // 처리할 내용
} finally {
    // 실행할 본문
}
```

# 다중 catch

- **하나의 try**에 대해 **여러 개의 catch**를 적용하는 것
- 위에 작성한 catch부터 아래에 작성한 catch까지 순서대로 적용

# 다중 catch

```
try {
    // 시도할 코드
} catch ([예외 클래스명1] [변수명]) {
    // 처리할 내용
} catch ([예외 클래스명2] [변수명]) {
    // 처리할 내용
} catch ([예외 클래스명3] [변수명]) {
    // 처리할 내용
}
```

# multi catch

- **하나의 catch**에 **여러 개의 예외**를 적용하는 것
- |(or) 연산자로 작성

# multi catch

```
try {
    // 시도할 코드
} catch ([예외 클래스명1] | [예외 클래스명2] [변수명]) {
    // 처리할 내용
}
```

# try-with-resources

- try 블록의 실행이 끝난 후 **자원을 자동으로 반납**하는 구문
- **try + finally**
- try 키워드와 블록 사이에 반드시 **자원의 선언과 초기화**를 해줘야 함

# try-with-resources

```
try ([자원을 갖는 클래스명] [변수명] = new [생성자]) {
    // 시도할 코드
}
```

# throws

- 예외가 발생한 경우, **예외를 직접 처리하지 않고 다른 곳으로 예외의 처리를 넘길 때** 사용하는 키워드
- 메서드 시그니처 마지막에 작성
- 예외 처리를 넘길 경우, **해당 메서드를 호출한 곳에서 처리**
- **main() 메서드**에서 예외 처리를 넘기면 **JVM**이 처리

# throws

```
메서드명 () throws [예외 클래스명] {
    // 메서드 본문
}
```

# throw

- **강제로 예외를 발생**시킬 때 사용하는 키워드
- **예외 객체를 생성** 가능
- 예외를 발생시키면 반드시 처리해줘야 함

# throws

```
메서드명 () throws [예외 클래스명] {
    throw new [예외 클래스 생성자];
}
```

# 사용자 정의 예외

- **예외 클래스를 상속**받도록 사용자가 작성한 임의의 클래스
- throw 키워드로 발생시키고 처리
- 보통 Exception 클래스를 상속받고 오버로딩 생성자를 이용

# 사용자 정의 예외

```
class [클래스명] extends [예외 클래스명] {
    public [생성자] {
        super("예외 처리 메시지");
    }
}
```