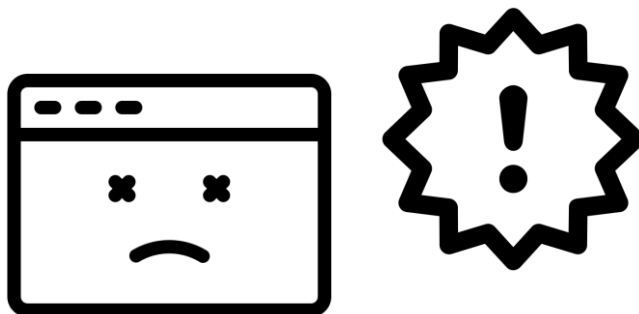


JAVA Programming

예외처리

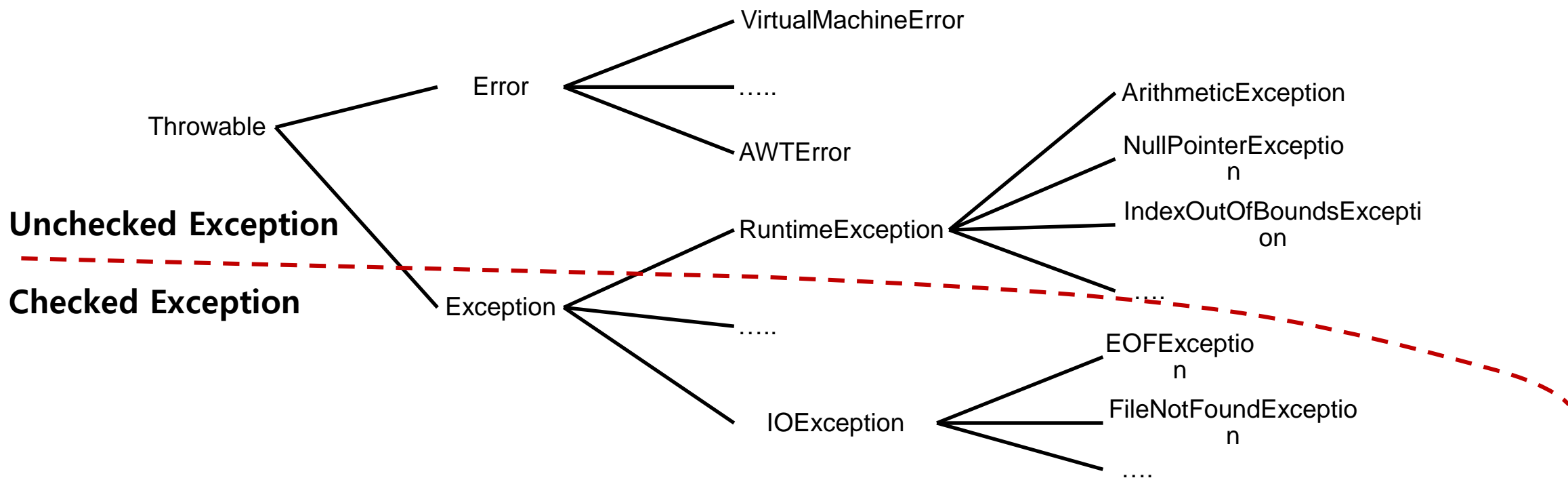
오류와 예외

- 시스템 상에서 프로그램에 심각한 문제가 발행해서 실행중인 프로그램이 영향을 받는 것은 오류와 예외로 구분할 수 있음
- 오류(Error)
시스템 상에서 프로그램에 심각한 문제를 발생하여 실행중인 프로그램이 종료되는 것
- 예외(Exception)
오류와 마찬가지로 비정상적으로 종료시키지만 미리 예측하고 **처리할 수 있는 미약한 오류**



예외 클래스 계층 구조

- Exception과 Error 클래스 모두 Throwable 클래스의 자손이다.
- 예외 클래스들의 최상위 클래스는 Exception 클래스이며 예외처리를 해야하는 Checked Exception과 해주지 않아도 되는 Unchecked Exception으로 나뉜다.



예외처리

- 예외는 예외처리를 통해 코드의 흐름을 컨트롤 가능
- 예외 처리 방법

1. throws로 위임

(Exception 처리를 호출한 메소드에게 위임)

메소드 선언 시 **throws** ExceptionName문을 추가하여 호출한 상위 메소드에게 처리를 위임

2. try-catch로 처리

(Exception이 발생한 곳에서 직접 처리)

try : exception 발생할 가능성이 있는 코드를 안에 기술

catch : try 구문에서 exception 발생 시 해당하는 exception에 대한 처리 기술

여러 개의 exception처리가 가능하나 exception간의 상속 관계 고려

finally : exception 발생 여부와 관계없이 꼭 처리해야 하는 로직 기술

중간에 return문을 만나도 finally구문은 실행되지만

System.exit();를 만나면 무조건 프로그램 종료

주로 java.io나 java.sql 패키지의 메소드 처리 시 이용

예외처리

- Throws로 예외 던지기

```
public static void main(String[] args) {  
    ThrowsTest tTest = new ThrowsTest();  
  
    try {  
        tTest.methodA();  
    } catch (IOException e) {  
        e.printStackTrace();  
    }  
}
```

```
public void methodA() throws  
IOException {  
    methodB();  
}
```

```
public void methodB() throws  
IOException {  
    methodC();  
}
```

```
public void methodC() throws  
IOException {  
    throw new IOException();  
    //Exception 발생  
}
```

예외처리 방법

- try~catch로 예외 처리

```
public void method() {  
    BufferedReader br = null;  
    try {  
        br = new BufferedReader(new FileReader("C:/data/text.txt"));  
        String s;  
        while((s = br.readLine()) != null) {  
            System.out.println(s);  
        }  
    } catch(FileNotFoundException e) {  
        System.out.println("파일이 없습니다.");  
    } catch(IOException e) {  
        e.printStackTrace();  
    }  
}
```

사용자 정의 예외

- Exception 클래스를 상속받아 예외 클래스를 작성하는 것

```
public class UserException extends Exception{  
    public UserException() {}  
    public UserException(String msg) {  
        super(msg);  
    }  
}
```

```
public class UserExceptionTest {  
    public void method() throws UserException{  
        throw new UserException("사용자정의 예외발생");  
    }  
}
```

```
public class Run {  
    public static void main(String[] args) {  
        UserExceptionTest uet = new UserExceptionTest();  
        try {  
            uet.method();  
        } catch (UserException e) {  
            e.printStackTrace();  
        }  
    }  
}
```

```
com.greedy.section02.userexception.Application1
```

```
com.greedy.section02.userexception.Application2
```

```
com.greedy.section02.userexception.Application3
```

RuntimeException 후손 클래스

- **ArithmeticException**
0으로 나누는 경우 발생
if문으로 나누는 수가 0인지 검사
- **ArrayIndexOutOfBoundsException**
배열의 index범위를 넘어서 참조하는 경우
배열명.length를 사용하여 배열의 범위 확인
- **NullPointerException**
Null인 참조 변수로 객체 멤버 시도 시 발생
객체 사용 전에 참조 변수가 null인지 확인
- **ClassCastException**
Cast연산자 사용 시 타입 오류
instanceof연산자로 객체 타입 확인 후 cast연산
- **NegativeArraySizeException**
배열 크기를 음수로 지정한 경우 발생
배열 크기를 0보다 크게 지정

예외처리 방법

- finally로 예외 처리

예외 처리 구문과 상관 없이 반드시 수행해야 하는 경우 작성
(보통 사용한 자원을 반납할 목적)

```
public static void main(String[] args) {  
    ThrowsTest tTest = new ThrowsTest();  
  
    try {  
        tTest.methodA();  
    } catch (IOException e) {  
        e.printStackTrace();  
    } finally {  
        System.out.println("프로그램 종료");  
    }  
}
```

예외처리 방법

- try~with~resource로 예외 처리

자바7에서 추가된 기능으로 finally에서 작성했던 close처리를 try문에서 자동으로 close처리

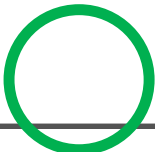
```
try (BufferedReader br = new BufferedReader(new FileReader("C:/data/text.txt"))){
    String s;
    while((s = br.readLine()) != null) {
        System.out.println(s);
    }
} catch(FileNotFoundException e) {
    System.out.println("파일이 없습니다.");
} catch(IOException e) {
    e.printStackTrace();
} catch(Exception e) {
    e.printStackTrace();
}
```

Exception과 오버라이딩


- 오버라이딩 시 throws하는 Exception의 개수와 상관없이 같거나 후손 범위여야 함

```
public class Parent {  
    public void method() throws IOException{  
        . . .  
    }  
}
```

```
public class Child1 extends Parent{  
    @Override  
    public void method() throws EOFException {  
        . . .  
    }  
}
```



```
public class Child2 extends Parent{  
    @Override  
    public void method() throws Exception {  
        . . .  
    }  
}
```



학습점검

- ✓ 오류와 예외를 구분할 수 있다.
- ✓ 예외처리의 목적을 이해할 수 있다.
- ✓ Throws를 활용해 예외처리를 할 수 있다.
- ✓ try-catch를 활용해 예외처리를 할 수 있다.
- ✓ finally를 추가할 수 있다.
- ✓ try-with-resource를 활용해 예외처리를 할 수 있다.
- ✓ 상속 관계에서의 예외처리에 대해 이해할 수 있다.