12장 예외처리

자바의 에러

◈ 컴파일 타임 에러(Compile-Time Error)

자바의 문법적 오류로 컴파일이 되지 않는 구문상의 오류

◈ 실행 타임 에러(Run-Time Error)

컴파일은 되지만 실행이 되지 않는 로직(Logic) 상의 오류

보통 시스템 문제로 인해 발생하며, 프로그램 코드로 에러를 처리할 수 있는 방법은 없다.

자바의 예외

◈ 예외(Exception)

예외란 컴퓨터 시스템이 동작하는 도중에 예상하지 못한 오류가 발생하는 것

예외 상황이 발생하면 실행되고 있던 프로그램을 비정상적으로 종료시킨다.

따라서 예외 처리(Exception handling)를 통해 예외 상황을 처리할 수 있도록 코드의 흐름을 바꾼다.

◈ Checked Exception

컴파일러에 의해 체크가 되는 Exception

◈ Unchecked Exception

런타임 시에 체크되는 Exception

자바의 예외

◈ 에러와 예외 차이

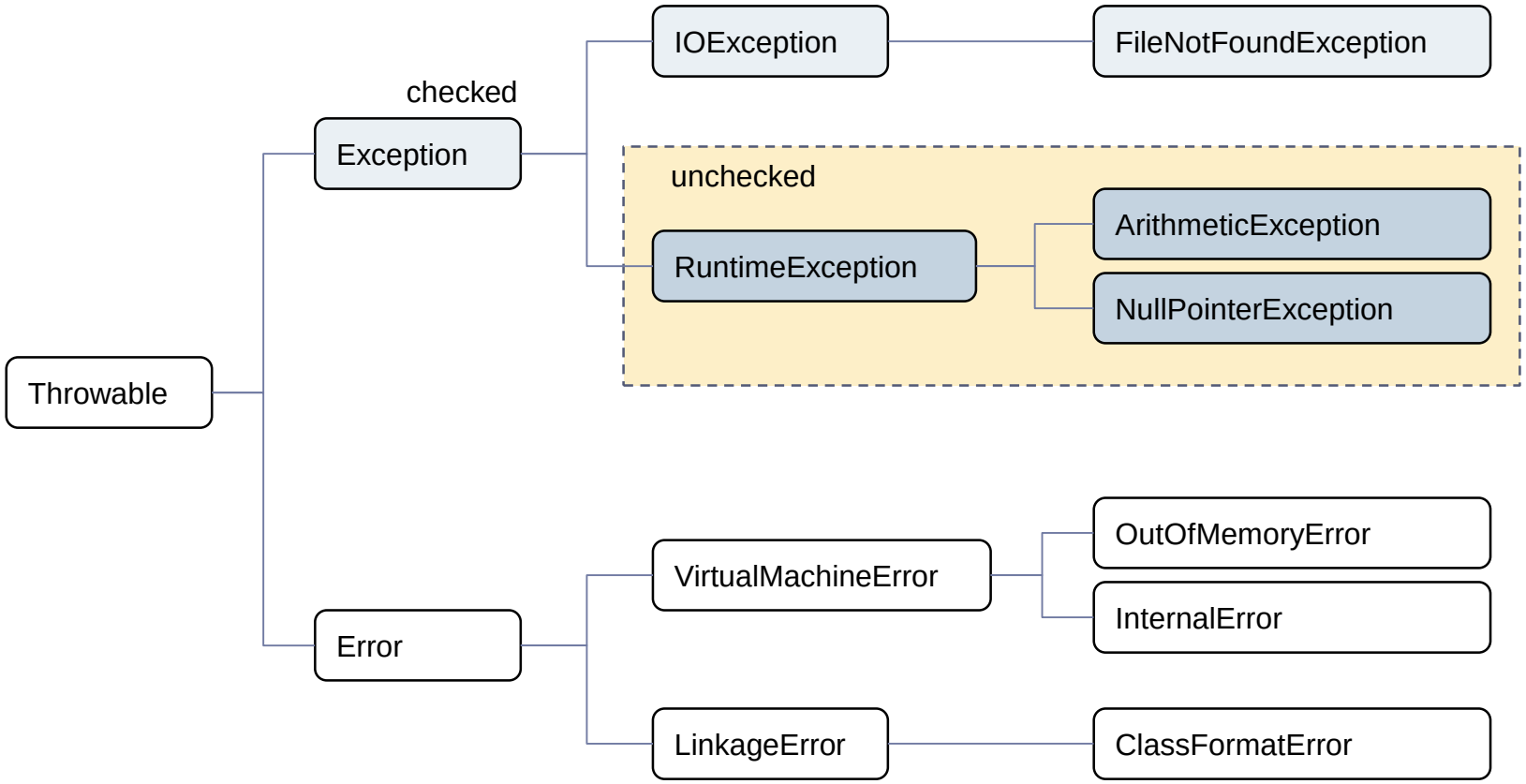
비교의 근거	오류	예외
기본	시스템 자원이 부족하여 오류가 발생했습니다.	코드로 인해 예외가 발생했습니다.
회복	오류는 복구 할 수 없습니다.	예외는 복구 가능합니다.
키워드	프로그램 코드에서 오류를 처리 할 수있는 방법은 없습니다.	예외는 3 개의 키워드 "try", "catch"및 "throw"를 사용하여 처리됩니다.
결과	오류가 감지되면 프로그램이 비정상적으로 종료됩니다.	예외가 감지되면 throw 및 catch 키워드에 따라 예외가 발생합니다.
유형	오류는 검사되지 않은 유형으로 분류됩니다.	예외는 체크 된 또는 확인되지 않은 유형으로 분류됩니다.
꾸러미	Java에서 오류는 "java.lang.Error"패키지로 정의됩니다.	Java에서 예외는 "java.lang.Exception"에 정의됩니다.
예	OutOfMemory, StackOverFlow.	확인 된 예외 : NoSuchMethod, ClassNotFound. 체크되지 않는 예외 : NullPointer, IndexOutOfBounds.

- 에러는 코드를 수정해서 에러가 나지 않도록 해야 한다.

- 예외는 코드를 수정하지 않더라도 이에 대한 별도의 처리가 가능하다.

자바의 예외

◈ 예외 클래스들



자바의 예외처리

◈ 예외 처리(Exception handling)

try-catch-finally 구문 사용

```
public static void main(String[] args)  {
    int[] intArray = {1,2,3};

    try {
        System.out.println(intArray[3]);    // 에러가 발생할 수 있는 구문
    }catch(Exception e) {
        e.printStackTrace();    // 예외 상황에 대한 에러 메세지 출력
    }finally {
        System.out.println("소리질러~~!!");    // 예외가 발생하든 안하든 실행
    }
}
```

1. try 블록 : 구문이 실행되는 도중 예외가 발생하면, 그 즉시 catch 블록의 구문이 실행된다.
2. catch 블록 : try 블록에서 발생한 예외 코드나 예외 객체를 인수로 전달받아 그 처리를 담당한다.
3. finally 블록 : try 블록에서 예외가 발생하든 안하든 맨 마지막에 무조건 실행되는 구문을 적는다.

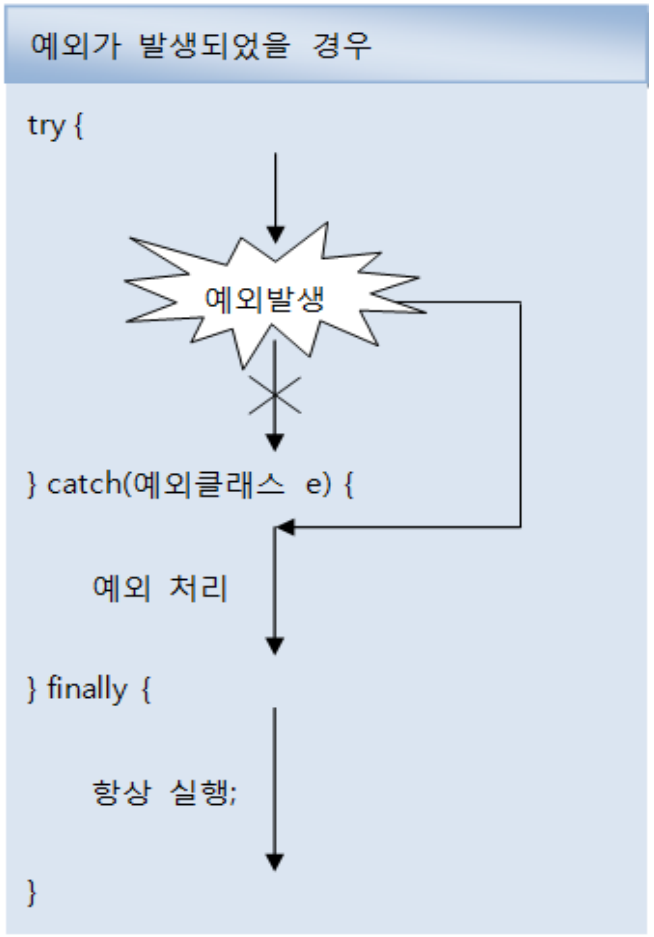
자바의 예외처리

◈ 예외 처리(Exception handling)

정상 실행 되었을 경우

```
try {
        예외 발생 가능 코드
} catch(예외클래스 e) {
        예외 처리
} finally {
        항상 실행;
}
```

예외가 발생되었을 경우



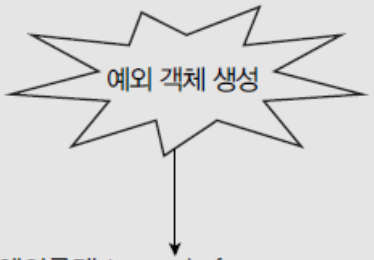
자바의 예외처리

◈ printStackTrace()

예외 발생 코드에 대해 추적한 내용을 모두 콘솔에 출력

프로그램 테스트하면서 오류 찾을 때 유용하게 활용

```
try {
```




```
} catch(예외클래스    e) {
    //예외가 가지고 있는 Message 얻기
    String message = e.getMessage();

    //예외의 발생 경로를 추적
    e.printStackTrace();
}
```

자바의 예외처리

◈ getMessage()

예외를 발생시킬 때, 생성자 매개값으로 입력했던 메시지 리턴

printStackTrace() 에 포함되어 있다.

```
throw new BizException("예외 메시지");
```

자바의 예외처리

◈ 예외 처리(Exception handling)

다중 catch 구문 사용

```
public static void main(String[] args)  {
    int[] intArray = {1,2,3};
    int numA = 12;
    int numB = 0;

    try {
        System.out.println(numA/numB);
        System.out.println(intArray[3]);
    }catch(ArithmeticException e) {
        e.printStackTrace();    // numA/numB 의 예외 상황
    }catch(IndexOutOfBoundsException e) {
        e.printStackTrace();    // intArray[3] 의 예외 상황
    }catch(Exception e) {
        e.printStackTrace();    // 그 외 예상하지 못한 예외가 발생한 경우
    }finally {
        System.out.println("소리질러~~!!");
    }
}
```

```
catch(ArithmeticException | IndexOutOfBoundsException e)
```

로 묶어서 사용 가능

자바의 예외처리

◈ 예외 처리(Exception handling)

throw 구문 사용

보통 개발자가 별도의 예외 처리를 하기 위해 사용자 정의 예외 클래스를 만들어서 사용

◈ 사용자 정의 예외 클래스

자바 표준 API에서 제공하지 않는 예외를 정의할 수 있다.

Exception 클래스를 상속받아 자신만의 새로운 예외 클래스를 정의하여 사용한다.

일반적으로 업무적 서비스와 관련하여 생성한다(BizException).

자바의 예외처리

◈ 사용자 정의 예외 클래스 선언 방법

```
public class XXXException extends [ Exception | RuntimeException ] {
    public XXXException() { }
    public XXXException(String message) { super(message); }
}
```

```
// 사용자 정의 예외
public class BizException extends RuntimeException {

    public BizException(String msg) {
        super(msg);
    }
    public BizException(Exception ex) {
        super(ex);
    }
}
```

```
// 입력받은 두 숫자를 나눈 값 리턴
public static int divide(int numA, int numB) throws BizException {
    if(numB == 0) {
        throw new BizException("numB는 0 이외의 숫자를 입력하세요");
    }else {
        return numA/numB;
    }
}
```

메소드를 사용하는 곳에서 try/catch구문 사용

```
try {
    int result = ExMethod.divide(5, 0);
    System.out.println(result);
}catch(BizException e) {
    System.out.println("예측한 BizException 에러발생");
}catch(Exception e) {
    System.out.println("예측 불가능한 에러");
}
```

자바의 예외처리

◈ 예외 처리(Exception handling)

throws 구문 사용

메서드 선언 부 끝에 throws를 붙이면 해당 메서드에서 예외처리를 하지 않고, 메서드를 호출한 곳에서 예외처리를 하도록 넘길 수 있다.

```
public static void main(String[] args)  {
    int numA = 12;
    int numB = 0;

    try {
        divide(numA, numB);
    }catch(ArithmeticException e) {
        System.out.println("0으로 나눌 수 없습니다.");
        e.printStackTrace();
    }
}

static void divide(int a, int b) throws ArithmeticException {
    System.out.println(a/b);
}
```

자바의 예외처리

◈ 예외 처리(Exception handling)

try-with-resource 구문 사용

각종 입출력 스트림, 서버 소켓, 채널 등과 같이 비정상적으로 종료되는 경우 문제가 발생하는 경우에 사용

예외 발생 여부와 관계 없이 리소스 객체의 close() 메서드를 실행하여 리소스를 닫는다.

리소스 객체는 java.lang.AutoCloseable 인터페이스를 구현하고 있어야 한다.

```
public static void main(String[] args) {
    try (Scanner scanner = new Scanner(System.in)) {
        System.out.println(scanner.nextLine());
    } catch (Exception e) {
        e.printStackTrace();
    }
}
```