SAMSUNG SW ACADEMY FOR YOUTH

함께가요 미래로!
Enabling People

삼성 청년 SW 아카데미

Java

SAMSUNG SW ACADEMY FOR YOUTH

Programming Language

객체지향 프로그래밍

- 예외처리

SAMSUNG SW ACADEMY FOR YOUTH

함께가요 미래로!
Enabling People

예외처리
(Exception Handling)

예외처리 (Exception Handling)

에러와 예외

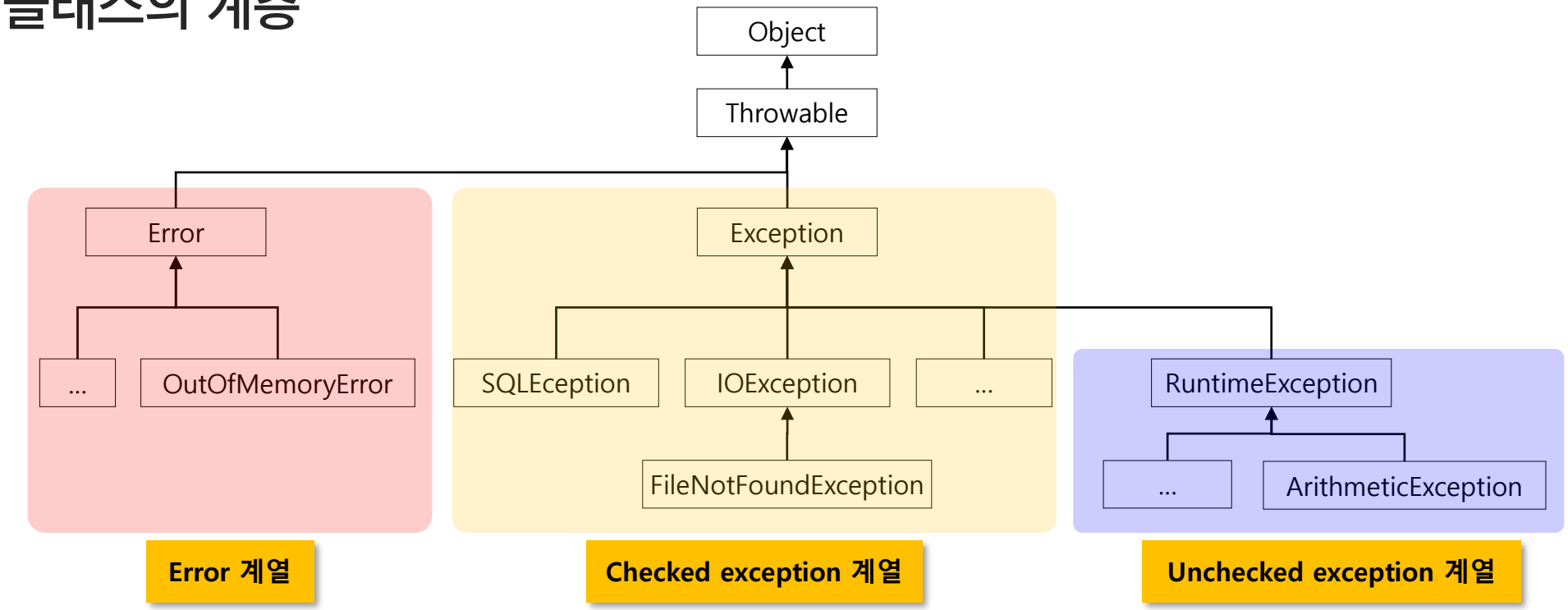
- 어떤 원인에 의해 오동작 하거나 비정상적으로 종료되는 경우
- 심각도에 따른 분류
  - Error
    - 메모리 부족, stack overflow 와 같이 일단 발생하면 복구할 수 없는 상황
    - 프로그램의 비 정상적 종료를 막을 수 없음 → 디버깅 필요
  - Exception
    - 읽으려는 파일이 없거나, 네트워크 연결이 안 되는 등 수습될 수 있는 비교적 상태가 약한 것들
    - 프로그램 코드에 의해 수습될 수 있는 상황
- 예외의 정의: An *exception* is an event, which occurs during the execution of a program, that disrupts the normal flow of the program's instructions.
- 에러의 정의: An Error is a subclass of Throwable that indicates serious problems that a reasonable application should not try to catch.

예외처리 (Exception Handling)

- 예외처리 (Exception Handling) 란
  - 예외 발생 시 프로그램의 비 정상 종료를 막고 정상적인 실행 상태를 유지하는 것
  - 예외의 감지 및 예외 발생 시 동작할 코드 작성 필요

예외처리 (Exception Handling)

예외 클래스의 계층



- Checked exception
  - 예외에 대한 대처 코드가 없으면 컴파일이 진행되지 않음
- Unchecked exception (RuntimeException의 하위 클래스)
  - 예외에 대한 대처 코드가 없더라도 컴파일은 진행됨

예외처리 (Exception Handling)

예외의 발생

```
public static void main(String[] args) {
  int[] nums = { 10 };
  System.out.println(nums[2]);
}
```

```
Exception in thread "main"
java.lang.ArrayIndexOutOfBoundsException: 2
```

예외처리 (Exception Handling)

- 예외 처리 키워드
  - 직접 처리
    - try
    - catch
    - finally
  - 간접 처리
    - throws
  - 사용자 정의 예외 발생시킬 때
    - throw

예외처리 (Exception Handling)

- try ~ catch 구문

```
try {
// 예외가 발생할 수 있는 코드
} catch (Exception e) {
// 예외가 발생했을 때 처리할 코드
}
```

- ex)

```
public static void main(String[] args) {

    int[] nums = { 10 };

    try {
        System.out.println(nums[2]);
    } catch (ArrayIndexOutOfBoundsException e) {
        System.out.println("배열의 크기 확인 필요 예외 발생");
    }
        System.out.println("프로그램을 종료합니다.");
}
```

Exception 객체의 정보 활용

- Throwable의 주요 메서드

메서드	설명
public String getMessage()	발생된 예외에 대한 구체적인 메시지를 반환한다.
public Throwable getCause()	예외의 원인이 되는 Throwable 객체 또는 null을 반환한다.
public void printStackTrace()	예외가 발생된 메서드가 호출되기까지의 메서드 호출 스택을 출력한다. 디버깅의 수단으로 주로 사용된다.

예외처리 (Exception Handling)

- try ~ catch 문에서의 흐름
  - try 블록에서 예외가 발생하면
  - JVM이 해당 Exception 클래스의 객체 생성 후 던짐(throw) : throw new XXException()
  - 던져진 exception을 처리할 수 있는 catch 블록에서 받은 후 처리 (적당한 catch 블록을 만나지 못하면 예외처리는 실패)
  - 정상적으로 처리되면 try-catch 블록을 벗어나 다음 문장 진행

  - try 블록에서 어떠한 예외도 발생하지 않는 경우
  - catch문을 거치지 않고 try-catch 블록의 다음 흐름 문장을 실행

예외처리 (Exception Handling)

다중 exception handling

- try 블록에서 여러 종류의 예외가 발생할 경우
- 하나의 try 블록에 여러 개의 catch 블록 추가 가능 (예외 종류별로 catch 블록 구성)

```
try {
  // exception이 발생할 만한 코드
} catch (XXException e) {
  // XXException 발생 시 처리 코드
} catch (YYException e) {
  // YYException 발생 시 처리 코드
} catch (Exception e) {
  // Exception 발생 시 처리 코드
}
```

CCException 발생

처리 가능?

```
try {
  // exception이 발생할 만한 코드
} catch (Exception e) {
  // Exception 발생 시 처리 코드
} catch (YYException e) {
  // YYException 발생 시 처리 코드
} catch (XXException e) {
  // XXException 발생 시 처리 코드
}
```

만약 이 순서라면?

- 다중 catch 문장 작성 순서 유의 사항
  - JVM이 던진 예외는 catch 문장을 찾을 때는 다형성이 적용됨
  - 상위 타입의 예외가 먼저 선언되는 경우 뒤에 등장하는 catch 블록은 동작할 기회가 없음
  - 상속 관계가 없는 경우는 무관
  - 상속 관계에서는 작은 범위(자식)에서 큰 범위(조상)순으로 정의

예외처리 (Exception Handling)

- try ~ catch ~ finally 구문을 이용한 예외 처리
  - finally는 예외 발생 여부와 상관 없이 언제나 실행
  - 중간에 return 을 만나는 경우도 finally 블록을 먼저 수행 후 return 실행

```
try {
  // exception이 발생할 만한 코드 - System 자원 사용
}catch(Exception e) {
  // XXException 발생 시 처리코드
}finally {
  // try block에서 접근했던 System자원의 안전한 원상복구
}
```

```
public static void main(String[] args) {
  int num = new Random().nextInt(2);
  try {
    System.out.println("code 1, num: " + num);
    int i = 1 / num;
    System.out.println("code 2 - 예외 없음");
    return;
  } catch (ArithmeticException e) {
    System.out.println("code 3 - exception handling 완료");
  } finally {
    System.out.println("code 4 - 언제나 실행");
  }
  System.out.println("code 5");
}
```

예외처리 (Exception Handling)

try ~ catch ~ finally 구문을 이용한 예외 처리

- finally를 이용한 자원 정리

```
class InstallApp {
  void copy() {
    System.out.println("파일 복사");
  }

  void install() throws Exception {
    System.out.println("설치");
    if (Math.random() > 0.5) {
      throw new Exception();
    }
  }

  void delete() {
    System.out.println("파일 삭제");
  }
}
```

```
InstallApp app = new InstallApp();
try {
  app.copy();
  app.install();
  app.delete();
} catch (Exception e) {
  app.delete();
  e.printStackTrace();
}
```

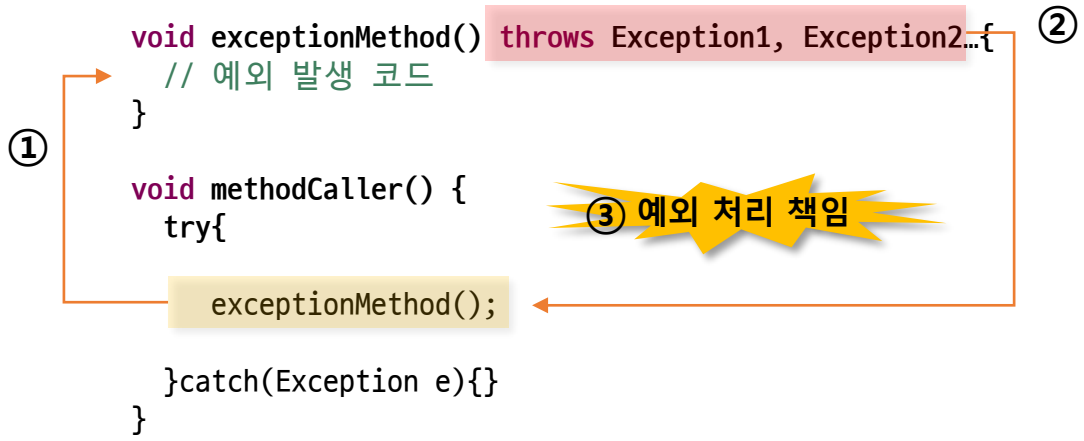
```
InstallApp app = new InstallApp();
try {
  app.copy();
  app.install();
} catch (Exception e) {
  e.printStackTrace();
} finally{
  app.delete();
}
System.out.println("설치 종료");
```

```
try {
  app.copy();
  app.install();
} catch (Exception e) {
  e.printStackTrace();
}
app.delete();
```

이렇게 작성하면???

예외처리 (Exception Handling)

- throws 키워드를 통한 처리 위임
  - method에서 처리해야 할 하나 이상의 예외를 호출한 곳으로 전달(처리 위임)
  - 예외가 없어지는 것이 아니라 단순히 전달됨
  - 예외를 전달받은 메서드는 다시 예외 처리의 책임 발생



  - 처리하려는 예외의 조상 타입으로 throws 처리가능

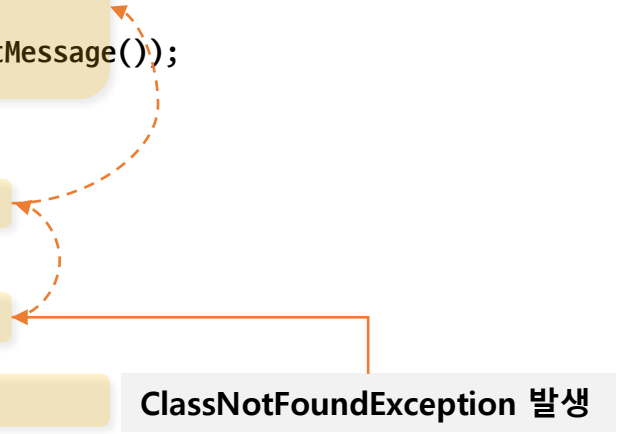
예외처리 (Exception Handling)

checked exception 과 throws

```
public static void main(String[] args) {
    CheckedThrowsTest et = new CheckedThrowsTest();
    try {
      et.method1();
    } catch (ClassNotFoundException e) {
      System.out.printf("exception handling: %s%n",e.getMessage());
    }
    System.out.println("프로그램 종료");
  }
  public void method1() throws ClassNotFoundException {
    method2();
  }
  public void method2() throws ClassNotFoundException {

    Class.forName("Some Class");

  }
```



- checked exception은 반드시 try ~ catch 또는 throws 필요
- 필요한 곳에서 try~catch 처리

예외처리 (Exception Handling)

runtime exception과 throws

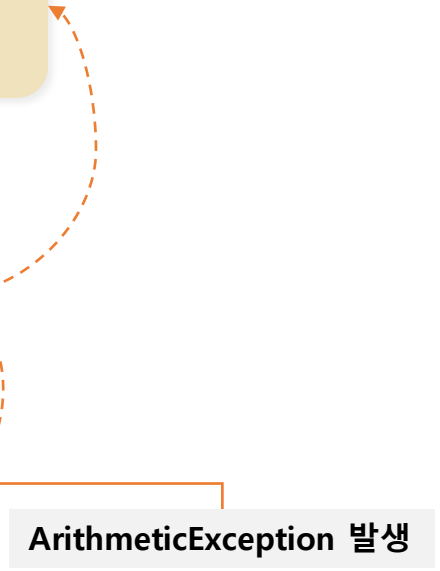
```
public static void main(String[] args) {
  RuntimeThrowsTest et = new RuntimeThrowsTest();
  try {
    et.method1();
  } catch (ArithmeticException e) {
    System.out.printf("예외 처리: %s%n",e.getMessage());
  }
  System.out.println("프로그램 종료");
}

public void method1() {
  method2();
}

public void method2() {

  int i = 1/0;

}
```

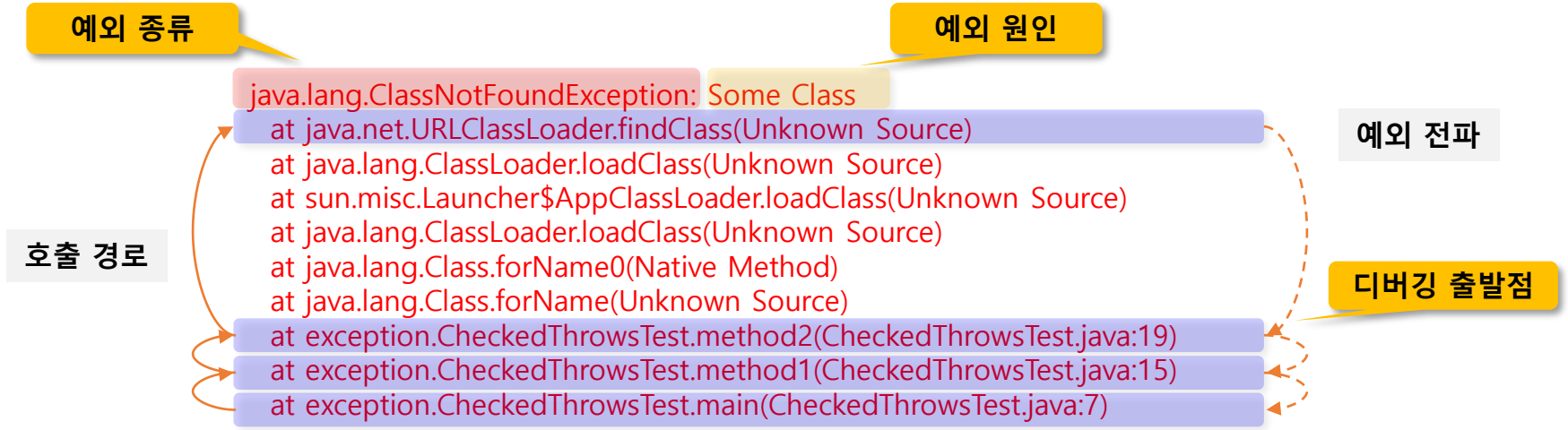


- runtime exception은 throws 하지 않아도 전달되지만
- 하지만 결국은 try~catch로 처리해야 함

예외처리 (Exception Handling)

로그 분석과 예외의 추적

- Throwable의 printStackTrace는 메서드 호출 스택 정보 조회 가능
  - 최초 호출 메서드에서부터 예외 발생 메서드 까지의 스택 정보 출력
- 꼭 확인해야할 정보
  - 어떤 예외인가? : 예외 종류
  - 예외 객체의 메시지는 무엇인가? : 예외 원인
  - 어디서 발생했는가? : 디버깅 출발점
    - 직접 작성한 코드를 디버깅 대상으로 삼을 것
    - 참조하는 라이브러리는 과감히 건너 뛰기



예외처리 (Exception Handling)

메서드 재정의와 throws

- 메서드 재정의 시 조상클래스 메서드가 던지는 예외보다 부모 예외를 던질 수 없다.

```
class Parent{
  void methodA() throws IOException{}
  void methodB() throws ClassNotFoundException{}
}

public class OverridingTest extends Parent {

  @Override
  void methodA() throws FileNotFoundException {

  }

  @Override
  void methodB() throws Exception {

  }
}
```

잘못된 부분은 어디에 있을까??

예외처리 (Exception Handling)

사용자 정의 예외

- API에 정의된 exception이외에 필요에 따라 사용자 정의 예외 클래스 작성

- 대부분 Exception 또는 RuntimeException 클래스를 상속받아 작성
  - checked exception 활용 : 명시적 예외 처리 또는 throws 필요
    (코드는 복잡해지지만 처리, 누락 등 오류 발생 가능성은 down)
  - runtime exception 활용 : 묵시적 예외 처리 가능
    (코드가 간결해지지만 예외 처리, 누락 가능성 발생)

- 사용자 정의 예외를 만들어 처리하는 장점
  - 객체의 활용 : 필요한 추가정보, 기능 활용 가능
  - 코드의 재사용 : 동일한 상황에서 예외 객체 재사용 가능
  - throws 메커니즘의 이용 : 중간 호출 단계에서 return 불필요

예외처리 (Exception Handling)

- 영화를 찾지 못했을 때 사용할 TitleNotFoundException.java 를 구현한다.
- IMovieManager의 searchByTile() 에 throws TitleNotFoundException 을 추가 작성한다.
- MovieManagerImpl의 searchByTitle()에서 해당 인자로 넘겨받은 제목을 포함하는 영화가 존재하지 않으면 TitleNotFoundException을 던지도록 구현한다.
- MovieTest 에서 등록하지 않은 영화를 검색을 해보고 이를 확인해본다. (try / catch 사용)

SAMSUNG SW ACADEMY FOR YOUTH

함께가요 미래로!
Enabling People

다음 방송에서 만나요!

삼성 청년 SW 아카데미