예외

- 에러
- 예외 관련 API
- 예외 처리
- 사용자 정의 예외 발생

10-1

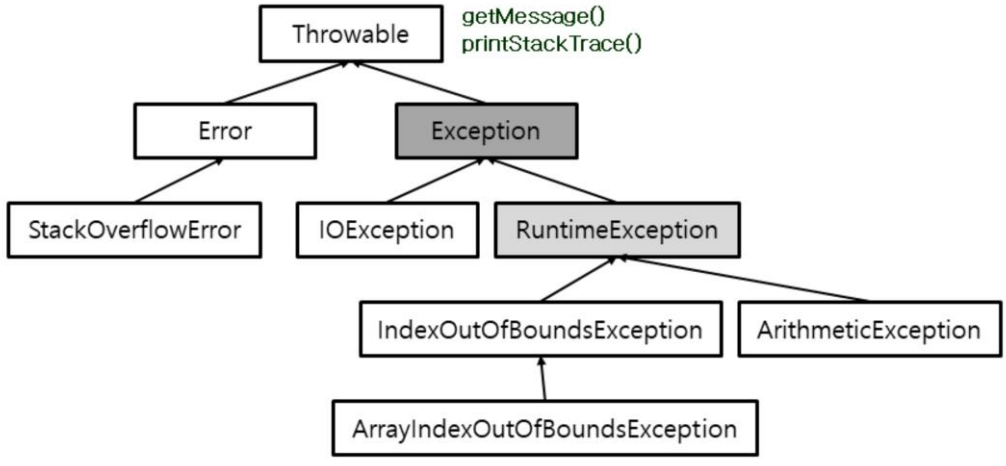
에러

- 에러
  - 프로그램 개발 과정에서 발생할 수 있는 기능, 비기능적 문제
- 에러 종류
  - 문법 에러
    - 주로 컴파일 과정에서 발생되고 반듯이 해결
  - 실행 오류
    - Error : 시스템 문제등 VM내 심각한 오류
    - Exception : 프로그램 메시지로 처리할 수 있는 가벼운 오류

10-2

예외 관련 API

- 예외 관련 패키지 – java.lang.*
- Throwable – Error, Exception의 공통 동작 정의



10-3

예외 처리

- 배열과 같이 클래스가 없는 객체는 실행해야 예외 종류를 확인 할 수 있다.
- 예외 발생시 처리를 하지 않으면 시스템은 비 정상적으로 종료 된다.

```
int [] nums={100, 200, 300};

for (int i = 0; i <= nums.length; i++) {
    System.out.println(nums[i]);
}
System.out.println("end");
```

```
100
200
300
Exception in thread "main" java.lang.ArrayIndexOutOfBoundsException: 3
        at kr.zeroand.java.exception.ExceptionTest_1.main(ExceptionTest_1.java:8)
```

예외 처리

- 예외 발생시 처리는 try/catch/finally
- 예외 처리시 발생된 예외나 부모 예외 클래스로 처리

```
int [] nums={100, 200, 300};

for (int i = 0; i <= nums.length; i++) {

    try{
        System.out.println(nums[i]);
    }catch(ArrayIndexOutOfBoundsException e){
        System.out.println(e.getMessage());
        e.printStackTrace();
    }finally{
        System.out.println("try or catch 후 마지막에 처리");
    }
}//for end
System.out.println("end");
```

10-5

예외 처리

- 여러 예외가 발생할 경우 catch시 상속관계를 고려
- 모든 예외는 Exception으로 한꺼번에 처리 가능
- NumberFormatException처럼 RuntimeException 계열의 예외는 컴파일 과정에서 체크하지 않는다.

```
String [] numbers={"123","456","abc","789"};
try{
    for (int i = 0; i < numbers.length; i++) {
        int num=Integer.parseInt(numbers[i])*10;
        System.out.println(num);
    }
}catch(NumberFormatException e){
    System.out.println(e.getMessage());
}catch(Exception e){
    e.printStackTrace();
}
System.out.println("end");
```

예외 처리

- CheckedException을 발생하는 객체는 try/catch or throws 하지 않으면 컴파일 오류가 발생한다.

```
3  public class ExceptionTest_5 {
4      public static void main(String[] args)  {
5          String s=null;
6          Class.forName("java.lang.String");
7          Class.forName("Employee");
8          System.out.println("end");
9      }
10
11 }
```

10-7

예외 처리

- main()에서 throws를 하면 컴파일 오류는 발생되지 않지만 Non CheckedException처럼 프로그램이 비 정상 종료한다.

```
public static void main(String[] args) throws ClassNotFoundException {
    String s=null;
    Class.forName("java.lang.String");
    Class.forName("Employee");
    System.out.println("end");
}
```

```
Exception in thread "main" java.lang.ClassNotFoundException: Employee
        at java.net.URLClassLoader$1.run(URLClassLoader.java:202)
        at java.security.AccessController.doPrivileged(Native Method)
        at java.net.URLClassLoader.findClass(URLClassLoader.java:190)
        at java.lang.ClassLoader.loadClass(ClassLoader.java:307)
        at sun.misc.Launcher$AppClassLoader.loadClass(Launcher.java:301)
        at java.lang.ClassLoader.loadClass(ClassLoader.java:248)
        at java.lang.Class.forName0(Native Method)
        at java.lang.Class.forName(Class.java:169)
        at kr.zeroand.java.exception.ExceptionTest_5.main(ExceptionTest_5.java:7)
```

- **ExceptionTest_6.java**

```
package kr.zeroand.java.exception;
public class ExceptionTest_6 {
    public void methodA() throws ClassNotFoundException{methodB();}
    public void methodB() throws ClassNotFoundException{methodC();}
    public void methodC() throws ClassNotFoundException{methodD();}
    public void methodD() throws ClassNotFoundException{
        Class.forName("Employee");
    }
    public static void main(String[] args)  {
        ExceptionTest_6 e1=new ExceptionTest_6();
        try {
            e1.methodA();
        } catch (ClassNotFoundException e) {
            e.printStackTrace();
        }
        System.out.println("end");
    }
}
```

사용자 정의 예외 발생

- 사용자가 정의한 메서드에 예외를 발생할 수 있다.
  - throw : 예외가 발생되는 위치
  - throws : 현재 메서드가 발생되는 예외를 선언적으로 제공
- 기존 예외 클래스 이용하여 발생

  throw new Exception("예외 발상");

```
public void setName(String name) throws Exception{
    if(name==null || name.length()<2)
    throw new Exception("이름은 2자 이상이어야 한다.");

    this.name=name;
}
```

사용자 정의 예외 발생

- 새로운 예외 클래스 생성 후 발생
  - 예외 클래스 작성
    - 예외 클래스 이름을 **XxxException**으로 정의
    - **Exception** 계열의 클래스 상속
    - 생성자에 **super(String)** 호출

```
class EmployeeNumberException extends Exception
{
        EmployeeNumberException(String message){
                super(message);
        }
        EmployeeNumberException(){
                this("직원번호는 6자리 숫자만");
        }
}
```

• 사용자 정의 예외 발생

```
/** 직원번호 : 꼭  6자리로 된 숫자 ==>EmployeeNumberException
 *  직원이름 : 내국인 꼭 2자리 이상  ==>Exception(이름길이 에러)
 * */

class Employee{
    private String name;
    private int employeeNumber;

    public Employee(int employeeNumber, String name)
                throws EmployeeNumberException, Exception{
        setEmployeeNumber(employeeNumber);
        setName(name);
    }

    public void setEmployeeNumber(int employeeNumber)
                throws EmployeeNumberException{

        if(!(employeeNumber>=100000 &&employeeNumber<1000000))
            throw new EmployeeNumberException("넘버는 6자리 숫자");

        this.employeeNumber=employeeNumber;
    }

    public void setName(String name) throws Exception{
        if(name==null || name.length()<2)
            throw new Exception("이름은 2자 이상이어야 한다.");

        this.name=name;
    }

    @Override
    public String toString() {
      return “name=“+name+", employeeNumber=“+employeeNumber;
    }
}
```

• 사용자 정의 예외 발생

```
class EmployeeNumberException extends Exception{
    EmployeeNumberException(String message){
        super(message);
    }
    EmployeeNumberException(){
        this("직원번호는 6자리 숫자만");
    }
}


public class ExceptionTest_7 {
    public static void main(String[] args) {
        Employee emp=null;

        try {
            emp=new Employee(1236, "홍길동");
        } catch (EmployeeNumberException e) {
            e.printStackTrace();
        } catch (Exception e) {
            e.printStackTrace();
        }
         System.out.println(emp);
    }
}
```

LAB

10-14