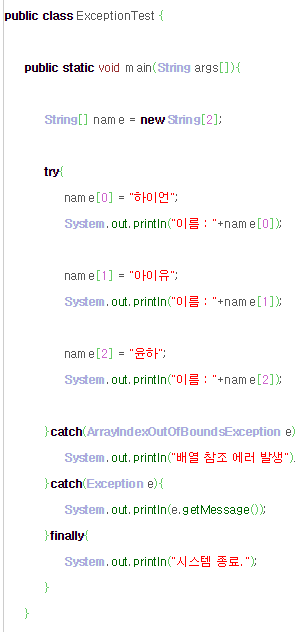
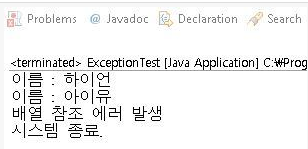
16장,17장

Try – catch- finally, throws , 사용자 정의 예외

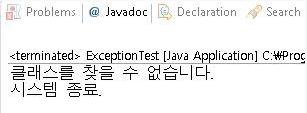
1.예외를 메소드 내에서 직접 처리(try-catch-finally)



If , if else 문과 비슷하게 예외를 타입 을 나누어 분류하는 방법이다. Finally 는 필수 블록은 아니지만 무조건 실행을 **가독성** 있게 보여준다.

2. 예외 발생한 메소드를 호출 한 곳으로 예외 객체를 넘기는 방법(throws)

Main함수에서 callDriver() 함수를 실행 시키면 callDriver() 함수에서는 oracle.jdbc~~ 클래스를 가져온다. 해당 클래스를 찾지 못하면 ClassnotfoundException이 발생 calldriver() 에서는 throws classNotfoundException 처리로 호출한 main함수로 예외를 넘긴다.

3. 사용자 정의 예외 생성 (throw)

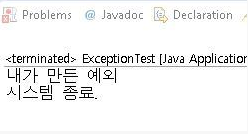
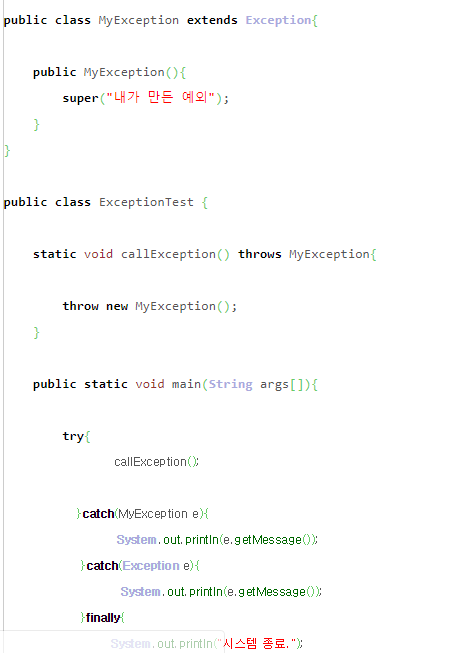
Java.lang.Exception 클래스를 상속받아 클래스를 정의한다.

-Class 에외 클래스 이름 extends Exception

자바 가상 머신은 프로그램 수행중에 예외가 발생하면 자동으로 예외 객체를 발생시킨 다음 catch 블록을 수행한다.

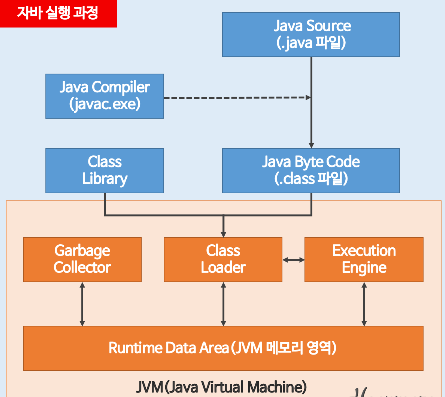
-throw new 예외 클래스 이름(매개 변수)

Throw를 이용한 예외 발생시에도 try-catch-finally 구문을 이용한 예외 처리를 하거나, throws를 이용하여 예외가 발생한 메소드를 호출한 다른 메소드로 넘기는 예외 처리 방법을 사용



Exception을 상속한 MyException을 만들고 MyException에 내가 만든 예외라는 메시지를 갖는다. main함수가 실행되면 callException() 함수를 호출하고 callException 함수는 new로 객체 생성후 main함수로 던진다.

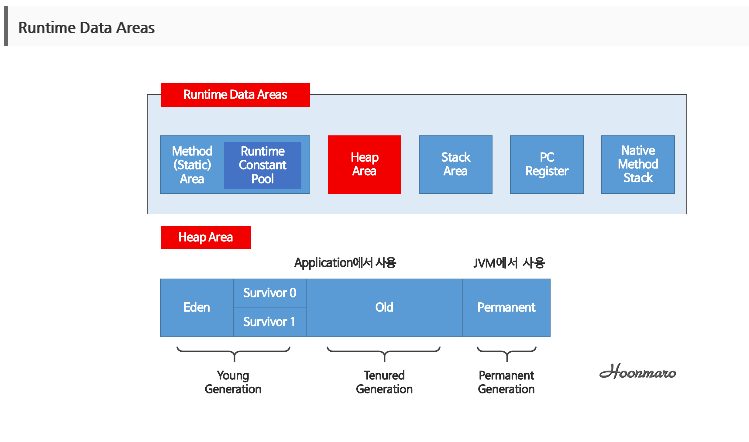
-17장 JVM구조



 Class Loader: JVM내로 클래스를 로드하고 링크를 통해 배치하는 작업을 수행하는 모듈로써 런타임시 동적으로 클래스를 로드한다.

 Execution Engine: Class Loader를 통해 JVM 내의 런타임 데이터 영역에 배치된 바이트 코드를 실행한다. 이 때, Execution Engine은 자바 바이트 코드를 명령어 단위로 읽어서 실행한다.

 Garbage Collector: JVM은 Garbage Collector를 통해 메모리 관리 기능을 자동으로 수행한다. 애플리케이션이 생성한 객체의 생존 여부를 판단하여 더 이상 사용되지 않는 객체를 해제하는 방식으로 메모리를 자동 관리한다.

 Runtime Data Areas: JVM이 운영체제 위에서 실행되면서 할당받는 메모리 영역이다. Class Loader에서 준비한 데이터들을 보관하는 저장소이다.  
  


* **Method (Static) Area**: JVM이 읽어들인 클래스와 인터페이스 대한 런타임 상수 풀, 멤버 변수(필드), 클래스 변수(Static 변수), 생성자와 메소드를 저장하는 공간이다.
* **Runtime Constant Pool**
  + 메소드 영역에 포함되지만 독자적 중요성이 있다.
  + 클래스 파일 constant\_pool 테이블에 해당하는 영역이다.
  + 클래스와 인터페이스 상수, 메소드와 필드에 대한 모든 레퍼런스를 저장한다.
  + JVM은 런타임 상수 풀을 통해 해당 메소드나 필드의 실제 메모리 상 주소를 찾아 참조한다
* **메소드 영역/런타임 상수 풀의 사용기간 및 스레드 공유 범위**
  + JVM 시작시 생성
  + 프로그램 종료 시까지
  + 명시적으로 null 선언 시
  + 구성 방식이나 GC 방법은 JVM 벤더마다 다를 수 있다.
  + 모든 스레드에서 공유한다.
* **Heap Area**
  + JVM이 관리하는 프로그램 상에서 데이터를 저장하기 위해 런타임 시 동적으로 할당하여 사용하는 영역이다.
  + New 연산자로 생성된 객체 또는 객체(인스턴스)와 배열을 저장한다.
  + 힙 영역에 생성된 객체와 배열은 스택 영역의 변수나 다른 객체의 필드에서 참조한다.
  + 참조하는 변수나 필드가 없다면 의미 없는 객체가 되어 GC의 대상이 된다.
  + 힙 영역의 사용기간 및 스레드 공유 범위
    - 객체가 더 이상 사용되지 않거나 명시적으로 null 선언 시
    - GC(Garbage Collection) 대상
    - 구성 방식이나 GC 방법은 JVM 벤더마다 다를 수 있다.
    - 모든 스레드에서 공유한다.
* **Stack Area**
  + 각 스레드마다 하나씩 존재하며, 스레드가 시작될 때 할당된다.
  + 메소드를 호출할 때마다 프레임(Frame)을 추가(push)하고 메소드가 종료되면 해당 프레임을 제거(pop)하는 동작을 수행한다.
  + 선입후출(FILO, First In Last Out) 구조로 push와 pop 기능 사용
  + 메소드 호출 시 생성되는 스레드 수행정보를 기록하는 Frame을 저장
  + 메소드 정보, 지역변수, 매개변수, 연산 중 발생하는 임시 데이터 저장
  + **기본(원시)타입 변수는 스택 영역에 직접 값을 가진다.**
  + **참조타임 변수는 힙 영역이나 메소드 영역의 객체 주소를 가진다.**
* **PC Register**
  + 현재 수행 중인 JVM 명령 주소를 갖는다.
  + 프로그램 실행은 CPU에서 인스트럭션(Instruction)을 수행.
  + CPU는 인스트럭션을 수행하는 동안 필요한 정보를 CPU 내 기억장치인 레지스터에 저장한다.
  + 연산 결곽값을 메모리에 전달하기 전 저장하는 CPU 내의 기억장치
* **Native Method Stack Area**
  + 자바 외 언어로 작성된 네이티브 코드를 위한 Stack이다.
  + 즉, JNI(Java Native Interface)를 통해 호출되는 C/C++ 등의 코드를 수행하기 위한 스택이다.
  + 네이티브 메소드의 매개변수, 지역변수 등을 바이트 코드로 저장한다.

**Heap Area**

* **Young Generation**: 이 영역은 자바 객체가 생성되자마자 저장되고, 생긴지 얼마 안되는 객체가 저장되는 공간이다. 시간이 지나 우선순위가 낮아지면 Old 영역으로 옮겨진다. 이 영역에서 객체가 사라질 때 Minor GC가 발생한다.
* **Old(Tenured) Generation**: Young Generation 영역에서 저장되었던 객체 중에 오래된 객체가 이동되어 저장되는 영역이다. 이 영역에서 객체가 사라질 때 Major GC(Full GC)가 발생한다.
* **Permanent Generation**: 클래스 로더에 의해 로든되는 클래스, 메소드 등에 대한 메타 정보가 저장되는 영역으로 JVM에 의해 사용된다. 리플렉션을 사용하여 동적으로 클래스가 로딩되는 경우에 사용된다. 내부적으로 리플렉션 기능을 자주 사용하는 Spring Framework를 이용할 경우 이 영역에 대한 고려가 필요하다.

-object 클래서는 모든 클래스의 최상위 클래스로써 간접적으로 상속을 받게 된다.

예를들어 class up{} {} // 여기서 object클래스 상속

Class Down extends Up{} // up클래스와 Object클래스를 상속

그럼 오브젝트 클래스를 살펴보자.

