

예외처리 및 제네릭 프로그래밍

예제 소스 코드는 파일과 연결되어 있습니다.
editplus(유료), notepad++(무료)와 같은 편집 도구를
미리 설치하여 PPT를 슬라이드 쇼로 진행할 때 소스
파일과 연결하여 보면 강의하실 때 편리합니다.

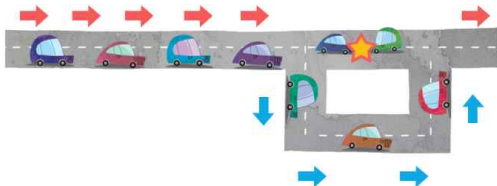
예외

■ 프로그램 오류의 종류

- 컴파일 오류 : 컴파일시 발생하는 오류(javac 소스.java 에서 발생하는 오류)
- 런타임 오류 : 실행할 때 발생하는 오류(java 소스.class)로 발생시 프로그램이 종료됨
- 논리적 오류 : 작성의도와 다르게 동작(프로그램이 종료되지는 않음)

■ Java의 런타임 오류

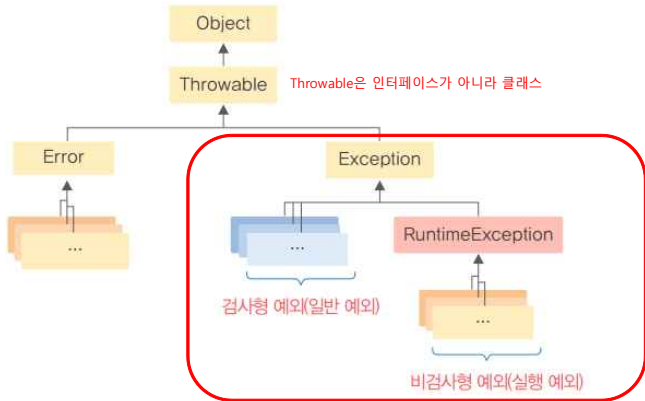
- 에러(error) : 개발자가 해결할 수 없는 치명적인 오류(코드로 수습 할 수 없는 심각한 오류)
- 예외(exception) : 개발자가 해결할 수 있는 오류(프로그램 코드로 수습 가능)
- 예외처리 : 에러는 어쩔 수 없지만, 예외가 발생하면 비정상적인 종료를 막고, 프로그램을 계속 진행할 수 있도록 우회 경로를 제공해서 정상적인 실행상태를 유지하는 것



예외

■ 종류

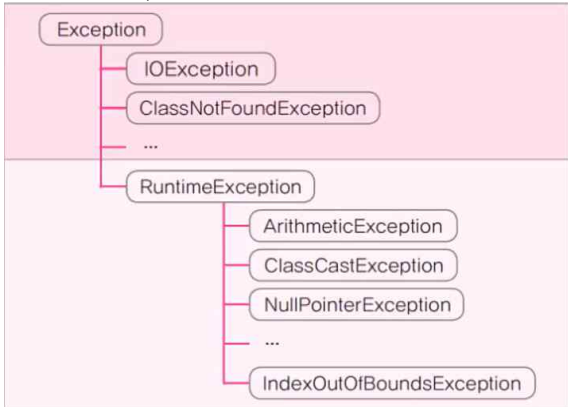
- 일반 예외와 실행 예외



예외

■ Exception(예외) 중심의 상속계층도

- 일반예외(Exception)클래스 : 사용자의 실수와 같은 외적인 요인에 의해 발생하는 예외
- 실행예외(RuntimeException)클래스 : 프로그래머의 실수로 발생하는 예외



예외

■ RuntimeException(실행 예외)

- 예외가 발생하면 JVM은 해당하는 실행 예외 객체를 생성
- 실행 예외는 컴파일러가 예외 처리 여부를 확인하지 않음. 따라서 개발자가 예외 처리 코드의 추가 여부를 결정
- 대표적인 실행 예외

실행 예외	발생 이유
ArithmeticException	0으로 나누기와 같은 부적절한 산술 연산을 수행할 때 발생한다.
IllegalArgumentException	메서드에 부적절한 인수를 전달할 때 발생한다.
IndexOutOfBoundsException	배열, 벡터 등에서 범위를 벗어난 인덱스를 사용할 때 발생한다.
NoSuchElementException	요구한 원소가 없을 때 발생한다.
NullPointerException	null 값을 가진 참조 변수에 접근할 때 발생한다.
NumberFormatException	숫자로 바꿀 수 없는 문자열을 숫자로 변환하려 할 때 발생한다.

- 예제
 - [sec01/Unchecked1Demo](#)
 - [sec01/Unchecked2Demo](#)

예외

■ 일반 예외

- 컴파일러는 발생할 가능성을 발견하면 컴파일 오류를 발생
- 개발자는 예외 처리 코드를 반드시 추가
- 대표적인 일반 예외 예

일반 예외	발생 이유
ClassNotFoundException	존재하지 않는 클래스를 사용하려고 할 때 발생한다.
InterruptedException	인터럽트되었을 때 발생한다.
NoSuchFieldException	클래스가 명시한 필드를 포함하지 않을 때 발생한다.
NoSuchMethodException	클래스가 명시한 메서드를 포함하지 않을 때 발생한다.
IOException	데이터 읽기 같은 입출력 문제가 있을 때 발생한다.

- 예제 : [sec01/CheckedDemo](#)

예외 처리(Exception handling) 방법

■ 예외처리 - Exception handling

- 정의 : 프로그램 실행 시 발생할 수 있는 예외의 발생에 대비한 코드를 작성하는 것
- 목적 : 프로그램의 비정상 종료를 막고, 정상적인 실행상태를 유지하는 것

■ 두 가지 방법

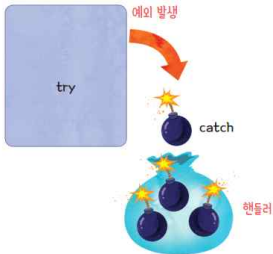
- 예외 잡아 처리하기
- 예외 떠넘기기

예외 처리(Exception handling) 방법

■ 예외 잡아 처리하기



(a) 일반적인 코드

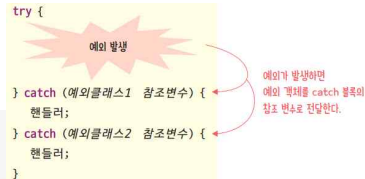
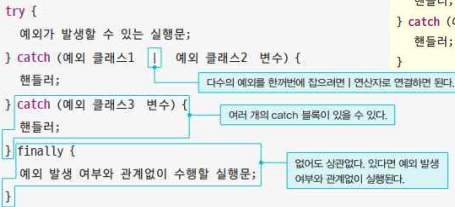


(b) try~catch 코드

예외 처리(Exception handling) 방법

■ 예외 잡아 처리하기

- catch 블록의 순서도 중요
- if문이나 for문과 달리 {} 를 생략할 수 없음



- 1) try블록 내에서 예외가 발생한 경우
 - 발생한 예외와 일치하는 catch 블록이 있는지 확인
 - 일치하는 catch블록내의 문장 수행 후 전체 try-catch문을 빠져나가서 그 다음 문장을 계속 수행.
 - 일치하는 catch블록을 찾지 못하면, 예외는 처리되지 못함
- 2) try블록 내에서 예외가 발생하지 않는 경우
 - catch 블록을 거치지 않고 전체 try-catch 문을 빠져나가서 수행을 계속

예외 처리(Exception handling) 방법

- 예외 잡아 처리하기 - exception 객체의 메소드
- `printStackTrace()` : 예외발생 당시의 호출스택(call stack)에 있었던 메서드의 정보와 예외 메시지를 화면에 출력
- `getMessage()` : 발생한 예외클래스의 인스턴스에 저장된 메시지를 얻을수 있음

```
try {  
    ...  
    System.out.println(0/0); // 예외발생  
} catch(ArithmeticException ae) {  
    ae.printStackTrace();  
    System.out.println(ae.getMessage());  
} catch (Exception e) {  
    ...  
}
```

예외 처리(Exception handling) 방법

- 멀티 catch 블록
- 내용이 같은 catch블록을 하나로 합친 것(JDK1.7부터)

```
try {  
    System.out.println(0/0);  
} catch(Exception1 e1) {  
    e1.printStackTrace();  
} catch (Exception2 e2) {  
    e2.printStackTrace();  
}
```



```
try {  
    System.out.println(0/0);  
} catch(Exception1 | Exception2 e) {  
    e.printStackTrace();  
}
```

■ 주의사항

- 부모자식 관계는 멀티 catch 블록안에 적을 수 없다.
catch(ParentException | ChildException e) // 적을 수 없다.
위 문장은 catch(ParentException e) 와 동일하기 때문이다.
- 두개의 예외객체에 서로다른 메소드는 catch블록내에서 호출할 수 없음

```
catch(Exception1 | Exception2 e) {  
    e.method!();  
}
```

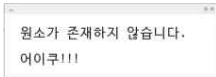
예외 처리(Exception handling) 방법

■ 예외 잡아 처리하기

- Throwable 클래스의 주요 메서드

메서드	설명
<code>public String getMessage()</code>	Throwable 객체의 자세한 메시지를 반환한다.
<code>public String toString()</code>	Throwable 객체의 간단한 메시지를 반환한다.
<code>public void printStackTrace()</code>	Throwable 객체와 추적 정보를 콘솔 뷰에 출력한다.

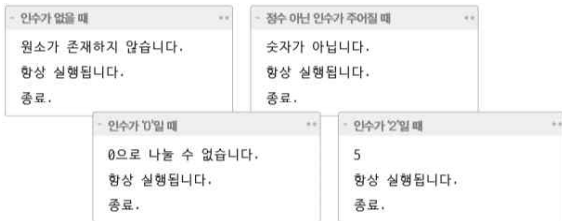
- 예제 : [sec02/TryCatch1Demo](#)



예외 처리(Exception handling) 방법

■ 예외 잡아 처리하기

- 예제 : [sec02/TryCatch2Demo](#)



- 예제 : [sec02/TryCatch3Demo](#)



예외 처리(Exception handling) 방법

■ 예외 잡아 처리하기

● try~with~resource 문

- try 블록에서 파일 등과 같은 리소스를 사용한다면 try 블록을 실행한 후 자원 반환 필요
- 리소스를 관리하는 코드를 추가하면 가독성도 떨어지고, 개발자도 번거롭다.
- JDK 7부터는 예외 발생 여부와 상관없이 사용한 리소스를 자동 반납하는 수단 제공. 단, 리소스는 AutoCloseable의 구현 객체

```
try (리소스) {  
    } catch ( ... ) {  
    }
```

- JDK 7과 8에서는 try()의 괄호 내부에서 자원 선언 필요. JDK 9부터는 try 블록 이전에 자원 선언 가능. 단, 선언된 자원 변수는 사실상 final이어야 함

● 예제 : [sec02/TryCatch4Demo](#)



예외 처리 방법

■ 예외 떠넘기기

- 메서드에서 발생한 예외를 내부에서 처리하기가 부담스러울 때는 throws 키워드를 사용해 예외를 상위 코드 블록으로 양도 가능



예외 처리 방법

■ 예외 떠넘기기

● 사용 방법

```
public void write(String filename)
    throws IOException, ReflectiveOperationException {
    // 파일 쓰기과 관련된 실행문 ...
}
```

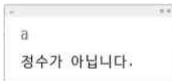
throws는 예외를 다른 메서드로 떠넘기는 키워드이다.

예외를 1개 이상 선언할 수 있다.

● 예제 : [sec02/ThrowsDemo](#)

● 자바 API 문서

- 많은 메서드가 예외를 발생시키고 상위 코드로 예외 처리를 떠넘긴다.
- 예를 들면,



```
public static void sleep(long millis, int nanos) throws InterruptedException
```


제네릭 타입

- 제네릭 : 포괄적인(구체적인의 반대) , 타입을 결정하지 않고 클래스를 설계
 - 필드선언할때 타입이 필요, 생성자의 매개변수 등에 타입이 들어간다.
 - 이러한 타입이 구체적이지 않고 포괄적인 타입으로 선언할 수 있다는 것이다.
 - 실제로 사용을 할때는 구체적인 타입이 결정이 되어야 하지만 설계할때는 구체적 타입을 언급하지 않고 사용할 때 구체적인 타입을 결정함
- 제네릭 타입의 의미
 - 하나의 코드를 다양한 타입의 객체에 재사용하는 객체 지향 기법
 - 결정되지 않은 타입을 파라미터를 가지는 클래스와 인터페이스를 제네릭 타입이라고 함
 - 클래스, 인터페이스, 메서드를 정의할 때 타입을 변수로 사용
- 제네릭 타입의 장점
 - 컴파일할 때 타입을 점검하기 때문에 실행 도중 발생할 오류 사전 방지
 - 타입 안정성을 제공
 - 불필요한 타입 변환이 없어 프로그램 성능 향상
 - 타입체크와 형변환을 생략할 수 있으므로 코드가 간결해짐



제네릭 타입

■ 필요성

- 자바는 다양한 종류의 객체를 관리하는 컬렉션이라는 자료구조를 제공
- 초기에는 Object 타입(모든 객체의 최상위 부모 클래스)의 컬렉션을 사용
- Object 타입의 컬렉션은 실행하기 전에는 어떤 객체인지?



- 예제(Object 타입)
 - [sec03/Beverage](#), [sec03/Beer](#), [sec03/Boricha](#), [sec03/object/Cup](#)
 - [sec03/GenericClass1Demo](#)



제네릭 타입

- Generics 예제
- 컴파일시 타입을 체크해 주는 기능(compile-time type check) - JDK1.5
- 예제를 컴파일하면? (성공), 실행하면? (ClassCastException 발생)

```
import java.util.ArrayList;
```

```
public class generic_test{  
    public static void main(String[] args) {  
        ArrayList list = new ArrayList();  
        list.add(10);  
        list.add("30");  
        list.add("abc");  
        // Integer i = (Integer)list.get(2); // 이부분을 추가하면 컴파일은 성공함  
        System.out.println(list);  
    }  
}  
} ==> [10, 30, abc] 가 출력됨
```

제네릭 타입

■ 제네릭 타입 선언

```
class 클래스이름<타입매개변수> {
```

```
    필드;
```

```
    메서드;
```

```
}
```

메서드나 필드에 필요한 타입을 타입 매개변수로 나타낸다.

- 타입 매개변수는 객체를 생성할 때 구체적인 타입으로 대체
- 전형적인 타입 매개변수

타입 매개변수	설명
E	원소(Element)
K	키(Key)
N	숫자(Number)
T	타입(Type)
V	값(Value)

- public class 클래스명<A,B, ...> (...)
- public interce 인터페이스명<A,B, ...> { ... }

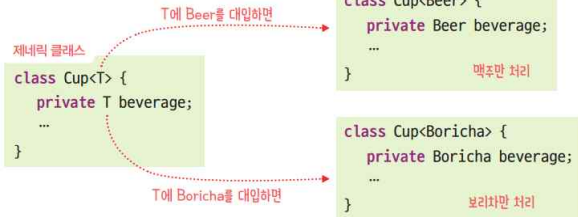
제네릭 타입

■ 제네릭 객체 생성

```
제네릭클래스 <적용할타입> 변수 = new 제네릭클래스<적용할타입>();
```

생략할 수 있다.

- <적용할타입>에서 적용할 타입을 생략할 경우 <>를 다이아몬드 연산자라고 함
- 제네릭 클래스의 적용



제네릭 타입

■ 제네릭 타입 응용

- 예제 : [sec03/generic/Cup](#), [sec03/GenericClass2Demo](#)



- 예제(2개 이상의 타입 매개변수)
 - [sec03/Entry.java](#)
 - [sec03/EntryDemo](#)

```
김선달 20  
기타 등등
```

■ Raw 타입의 필요성 및 의미

- 이전 버전과 호환성을 유지하려고 Raw 타입을 지원
- 제네릭 클래스를 Raw 타입으로 사용하면 타입 매개변수를 쓰지 않기 때문에 Object 타입이 적용
- 예제 : [sec03/GenericClass3Demo](#)

제네릭 상속 및 타입 한정

■ 제네릭 타입의 상속 관계

- 예를 들어

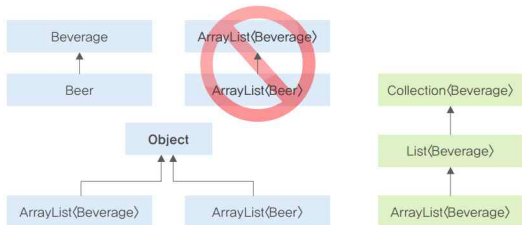
```
ArrayList<Beverage> list = new ArrayList<>();  
list.add(new Beer());           // OK  
list.add(new Boricha());       // OK
```

- 그러나 ArrayList<Beverage> 타입과 ArrayList<Beer>의 경우는 상속 관계가 없다.
- 예제 : [sec04/GenericInheritanceDemo](#)

제네릭 상속 및 타입 한정

■ 제네릭의 제약

- 기초 타입을 제네릭 인수로 사용 불가
- 정적 제네릭 타입 금지
- 제네릭 타입의 인스턴스화 금지. 즉, `new T()` 등 금지
- 제네릭 타입의 배열 생성 금지
- 실행 중에 제네릭 타입 점검 금지. 예를 들어, `a instanceof ArrayList<String>`
- 제네릭 클래스의 객체는 예외로 던지거나 잡을 수 없다
- 제네릭의 서브 타입 허용 않음



제네릭 메서드

■ 의미와 선언 방법

- 타입 매개변수를 사용하는 메서드
- 제네릭 클래스뿐만 아니라 일반 클래스의 멤버도 될 수 있음
- 제네릭 메서드를 정의할 때는 타입 매개변수를 반환 타입 앞에 위치

```
< 타입매개변수 > 반환타입 메서드이름(…) {  
    ...  
}
```

2개 이상의 타입 매개변수도 가능하다.

- 제네릭 메서드를 호출할 때는 구체적인 타입 생략 가능
- JDK 7과 JDK 8의 경우 익명 내부 클래스에서는 다이아몬드 연산자 사용 불허
- JDK 9부터는 익명 내부 클래스에서도 다이아몬드 연산자 사용 가능

제네릭 메서드

■ 예제

- 배열의 타입에 상관없이 모든 원소 출력
- [sec05/GenMethod1Demo](#)



```
1 2 3 4 5
H E L L O
5
```

제네릭 메서드

■ 제네릭 타입에 대한 범위 제한

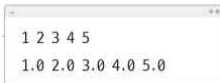
● 사용 방법

```
<T extends 특정클래스> 반환타입 메서드이름(...) { ... }  
<T extends 인터페이스> 반환타입 메서드이름(...) { ... }
```

부모가 인터페이스라도 extends를 사용한다.

● 예제

- [sec05/GenMethod2Demo](#)



```
1 2 3 4 5  
1.0 2.0 3.0 4.0 5.0
```

- [sec05/GenMethod3Demo](#)



```
3
```

제네릭 상속 및 타입 한정

■ 타입 한정

- 타입파라미터를 대체하는 구체적인 타입을 제한할 필요가 있다.
- 예> 숫자를 연산하는 제네릭메소드를 만들어야 하는 경우라면 대체타입으로 반드시 Number 또는 자식 클래스(Byte, Short, Integer, Long, Double)로 제한할 필요가 있다.
- 이처럼 모든 타입으로 대체할 수 없고, 특정 타입과 자식 또는 구현 관계에 있는 타입만 대체할 수 있는 타입 파라미터를 제한된 타입 파라미터라고 함.

```
<T extends 특정클래스> 반환타입 메서드이름(...) { ... }  
<T extends 인터페이스> 반환타입 메서드이름(...) { ... }
```

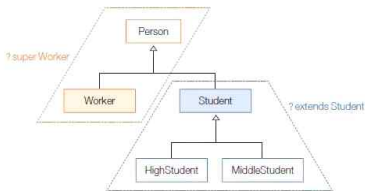
부모가 인터페이스라도 extends를 사용한다.

- 예제 : [sec04/bound/BoundedTypeDemo](#)

제네릭 상속 및 타입 한정

■ 와일드카드 타입 매개변수

- 제네릭타입을 인자값이나 리턴 타입으로 사용할 때 타입 매개변수로 ?(와일드카드)를 사용할 수 있다.



- Student 의 자식 클래스가 HighschoolStudent, MiddleschoolStudent인 경우 Student의 자식클래스만 가능하도록 한정하려면
 - 리턴타입 메소드명(제네릭타입<? extends Student> 변수) { ... }
- Worker의 부모 클래스인 Person 만 가능하도록 한정하려면
 - 리턴타입 메소드명(제네릭타입<? super Worker> 변수) { ... } 와 같이 선언할 수 있다.