Chapter 10

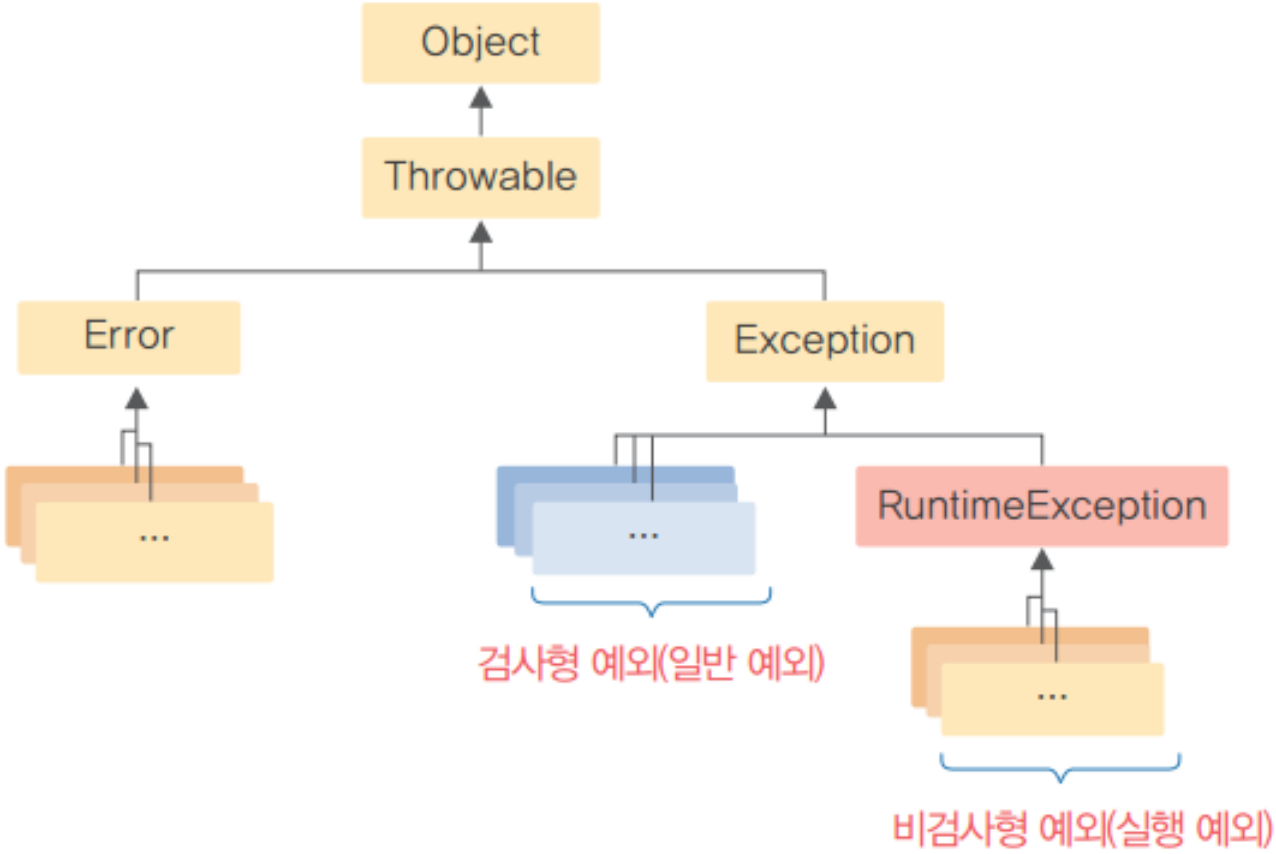
예외 처리와 제네릭 프로그래밍

Contents

예외의 개념

- 에러(error) : 개발자가 해결할 수 없는 치명적인 오류
- 예외(exception) : 개발자가 해결할 수 있는 오류
- 예외가 발생하면 비정상적인 종료를 막고, 프로그램을 계속 진행할 수 있도록 우회 경로를 제공하는 것이 바람 직

예외의 종류


Object
Throwable
Error
Exception
...
...
RuntimeException
...
검사형 예외(일반 예외)
비검사형 예외(실행 예외)

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 10-1]

```java
import java.util.StringTokenizer;

public class UnChecked1Demo {
    public static void main(String[] args) {
        String s = "Time is money";
        StringTokenizer st = new StringTokenizer(s);

        while (st.hasMoreTokens()) {
            System.out.print(st.nextToken() + "+");
        }
        System.out.print(st.nextToken());
    }
}
```

```
Exception in thread "main" java.util.NoSuchElementException
        at java.util.StringTokenizer.nextToken(Unknown Source)
        at sec01.UnChecked1Demo.main(UnChecked1Demo.java:13)
Time+is+money+
```

- [예제 10-2]

```
3  public class UnChecked2Demo {
4      public static void main(String[] args) {
5          int[] array = { 0, 1, 2 };
6
7          System.out.println(array[3]);
8      }
9  }
```

```
Exception in thread "main" java.lang.ArrayIndexOutOfBoundsException: 3
        at sec01.UnChecked2Demo.main(UnChecked2Demo.java:7)
```

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.

예외 처리 방법

- 예외 잡아 처리하기
- 예외 떠넘기기

예외 잡아 처리하기(1)

(a) 일반적인 코드

(b) try~catch 코드

예외 잡아 처리하기(2)

```
try {
    예외 발생
} catch (예외클래스1  참조변수) {
    핸들러;
} catch (예외클래스2  참조변수) {
    핸들러;
}
```

예외가 발생하면
예외 객체를 catch 블록의
참조 변수로 전달한다.

```
try {
   예외가 발생할 수 있는 실행문;
} catch (예외클래스1  |  예외클래스2  변수) {
   핸들러;
}
```

다수의 예외를 한꺼번에 잡으려면 | 연산자로 연결하면 된다.

예외 잡아 처리하기(3)

- Throwable 클래스의 주요 메서드

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

- [예제 10-4]

```
3  public class TryCatch1Demo {
4      public static void main(String[] args) {
5          int[] array = { 0, 1, 2 };
6          try {
7              System.out.println("마지막 원소 => " + array[3]);
8              System.out.println("첫 번째 원소 => " + array[0]);
9          } catch (ArrayIndexOutOfBoundsException e) {
10             System.out.println("원소가 존재하지 않습니다.");
11         }
12         System.out.println("어이쿠!!!");
13     }
14 }
```

```
원소가 존재하지 않습니다.
어이쿠!!!
```

- [예제 10-5]

```
3  public class TryCatch2Demo {
4      public static void main(String[] args) {
5          int dividend = 10;
6          try {
7              int divisor = Integer.parseInt(args[0]);
8              System.out.println(dividend / divisor);
9          } catch (ArrayIndexOutOfBoundsException e) {
10             System.out.println("원소가 존재하지 않습니다.");
11         } catch (NumberFormatException e) {
12             System.out.println("숫자가 아닙니다.");
13         } catch (ArithmeticException e) {
14             System.out.println("0으로 나눌 수 없습니다.");
15         } finally {
16             System.out.println("항상 실행됩니다.");
17         }
18         System.out.println("종료.");
19     }
20 }
```

실행 매개변수를 주지 않은 경우

```
원소가 존재하지 않습니다.
항상 실행됩니다.
종료.
```

실행 매개변수 a 를 준 경우

```
숫자가 아닙니다.
항상 실행됩니다.
종료.
```

실행 매개변수 0을 준 경우

```
0으로 나눌 수 없습니다.
항상 실행됩니다.
종료.
```

실행 매개변수 5를 준 경우

```
2
항상 실행됩니다.
종료.
```

- [예제 10-6]

```
3  public class TryCatch3Demo {
4      public static void main(String[] args) {
5          int[] array = { 0, 1, 2 };
6          try {
7              int x = array[3];
8          } catch (Exception e) {
9              System.out.println("어이쿠!!!");
10                 } catch (ArrayIndexOutOfBoundsException e) {
11                     System.out.println("원소가 존재하지 않습니다.");
12         }
13         System.out.println("종료.");
14     }
15 }
```

Unreachable catch block for ArrayIndexOutOfBoundsException. It is already handled by the catch block for Exception

예외 잡아 처리하기(4)

- try~with~resource 문
  - try 블록에서 파일 등 자원을 사용한다면 try 블록을 실행한 후 자원 반환 필요
  - 자원을 관리하는 코드를 추가하면 가독성도 떨어지고, 개발자도 번거롭다.
  - JDK 7부터는 예외 발생 여부와 상관없이 사용한 자원 자동 반납하는 수단 제공

```
try (자원) {
} catch ( … ) {
}
```

예외 떠넘기기(1)

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

예외 떠넘기기(2)

- 사용 방법

```
public void write(String filename)
      throws IOEception, ReflectiveOperationException {
   // 파일 쓰기와 관련된 실행문 …
}
```

예외를 1개 이상 선언할 수 있다.

throws는 예외를 다른 메서드로 떠넘기는 키워드이다.

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

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

- [예제 10-7]

```
3  import java.util.Scanner;
4
5  public class ThrowsDemo {
6      public static void main(String[] args) {
7          Scanner in = new Scanner(System.in);
8          try {
9              square(in.nextLine());
10         } catch (NumberFormatException e) {
11             System.out.println("정수가 아닙니다.");
12         }
13     }
14
15     private static void square(String s) throws NumberFormatException {
16         int n = Integer.parseInt(s);
17         System.out.println(n * n);
18     }
19 }
```

```
숫자가 아닙니다.
항상 실행됩니다.
종료.
```

제네릭의 개념과 필요성(1)

- 자바는 다양한 종류의 객체를 관리하는 컬렉션이라는 자료구조를 제공
- 초기에는 Object 타입의 컬렉션을 사용
- Object 타입의 컬렉션은 실행하기 전에는 어떤 객체가 담겨있는지 알 수 없음

제네릭의 개념과 필요성(2)

- 예제(Object 타입) : [예제 10-8]~[예제 10-10]

```
3  public class Beer {
4  }
```

```
3  public class Boricha {
4  }
```

```
03  public class Cup {
04     private Object beverage;
05
06     public Object getBeverage() {
07        return beverage;
08     }
09
10     public void setBeverage(Object beverage) {
11        this.beverage = beverage;
12     }
13  }
```

모든 종류의 음료수 객체를 Cup 객체에 담을 수 있도록 Object 타입을 사용한다.

제네릭의 개념과 필요성(3)

- 예제(Object 타입) : [예제 10-11]

```
03  public class GenericClass1Demo {
04    public static void main(String[] args) {
05       Cup c = new Cup();
06
07       c.setBeverage(new Boricha());
08
09       c.setBeverage(new Beer());
10
11       Beer b1 = (Beer) c.getBeverage();
12
13       Boricha b2 = (Boricha) c.getBeverage();
14    }
15  }
```

Cup 객체에 모든 객체를 담을 수 있으므로 Boricha 객체도 Beer 객체도 담을 수 있다. Cup 객체에는 Beer 객체만 남아 있다.

getBeverage( ) 메서드는 Object 타입을 반환하므로 Beer 타입으로 변환해야 한다.

Cup에 있는 Beer 객체를 Boricha 타입으로 변환하므로 실행 오류가 발생한다.

```
Exception in thread "main" java.lang.ClassCastException: sec03.Beer cannot be cast to sec03.Boricha
        at sec03.GenericClass1Demo.main(GenericClass1Demo.java:15)
```

제네릭의 개념과 필요성(4)

- Generic 타입
  - 하나의 코드를 다양한 타입의 객체에 재사용하는 객체 지향 기법
  - 클래스, 인터페이스, 메서드를 정의할 때 타입을 변수로 사용

- Generic 타입의 장점
  - 컴파일할 때 타입을 점검하기 때문에 실행 도중 발생할 오류 사전 방지
  - 불필요한 타입 변환이 없어 프로그램 성능 향상

제네릭 클래스와 인터페이스의 사용(1)

- 선언

```
class 클래스이름<타입매개변수> {
    필드;
    메서드;
}
```

메서드나 필드에 필요한 타입을 타입 매개변수로 나타낸다.

- 타입 매개변수는 객체를 생성할 때 구체적인 타입으로 대체

- 전형적인 타입 매개변수

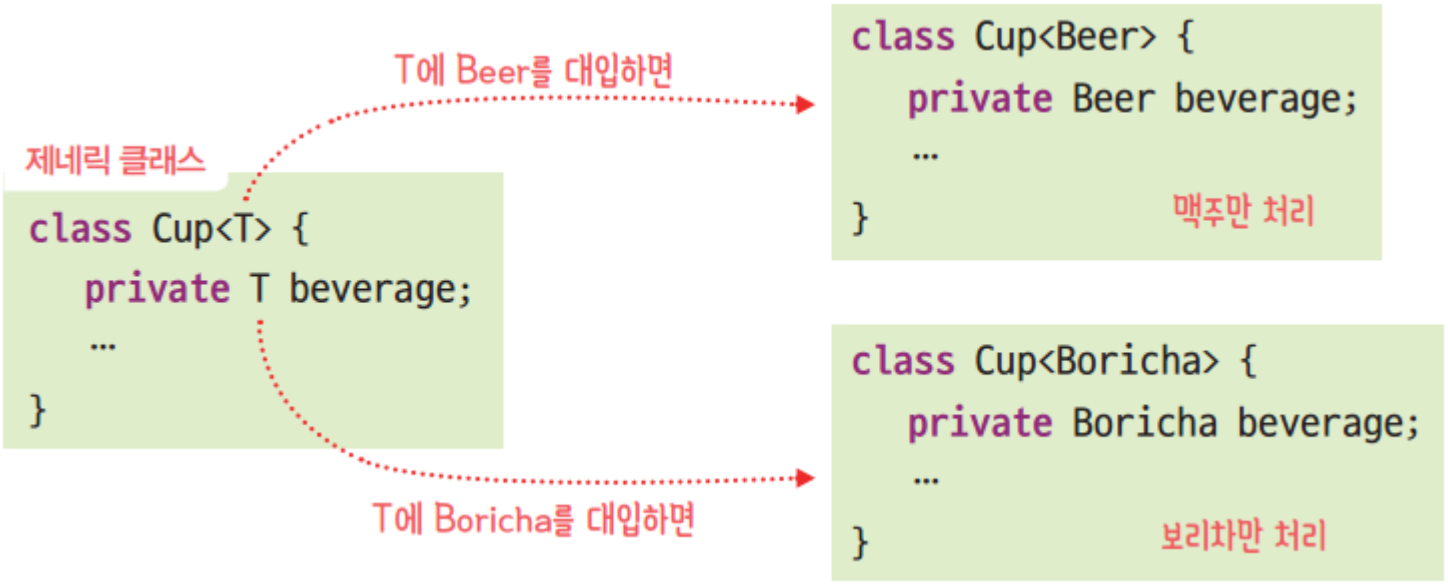
타입 매개변수	설명
E	원소(Element)
K	키(Key)
N	숫자(Number)
T	타입(Type)
V	값(Value)

제네릭 클래스와 인터페이스의 사용(2)

- 객체 생성

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

생략할 수 있다.



제네릭 클래스와 인터페이스의 사용(3)

- [예제 10-12]

```
02
03  public class Cup<T> {
04     private T beverage;
05
06     public T getBeverage() {
07        return beverage;
08     }
09
10     public void setBeverage(T beverage) {
11        this.beverage = beverage;
12     }
13  }
```

타입 매개변수를 명시한다.

제네릭 클래스와 인터페이스의 사용(4)

- [예제 10-13]

```
03  public class GenericClass2Demo {
04    public static void main(String[] args) {
05       Cup<Boricha> c = new Cup<Boricha>();
06
07       c.setBeverage(new Boricha());
08
09       // c.setBeverage(new Beer());
10
11       // Beer b = c.getBeverage();
12
13       Boricha b = c.getBeverage();
14    }
15  }
```

Boricha 타입의 Cup 객체를 생성한다.

Boricha 타입의 Cup 객체이기 때문에 Beer 타입의 객체를 담을 수 없다.

Cup 객체에 있는 Boricha 객체를 Beer 타입 매개변수에 대입할 수 없다.

Boricha 객체가 반환되므로 타입 변환이 필요 없다.

제네릭과 Raw 타입

- 이전 버전과 호환성을 유지하려고 Raw 타입을 지원
- 제네릭 클래스를 Raw 타입으로 사용하면 타입 매개변수를 쓰지 않기 때문에 Object 타입이 적용
- [예제 10-14]

```
05      Cup c1 = new Cup();
06
07      c1.setBeverage(new Beer());
08
09      // Beer beer = c1.getBeverage();
10      Beer beer = (Beer) c1.getBeverage();
```

구체적인 타입이 없으므로 Raw 타입의 제네릭 클래스를 사용한다.

Raw 타입의 Cup 객체이므로 어떤 타입의 객체든 추가할 수 있다.

어떤 타입이 반환되는지 알 수 없으므로 타입 변환이 필요하다.

2개 이상의 타입 매개변수(1)

- [예제 10-15]

```
01  public class Entry<K, V> {
02     private K key;
03     private V value;
04
05     public Entry(K key, V value) {
06        this.key = key;
07        this.value = value;
08     }
09
10     public K getKey() {
11     return key;
12     }
13
14     public V getValue() {
15        return value;
16     }
17  }
```

변수 key와 value의 타입은 각각 K와 V이다.

2개 이상의 타입 매개변수(2)

- [예제 10-16]

```
01  public class EntryDemo {
02    public static void main(String[] args) {
03       Entry<String, Integer> e1 = new Entry<>("김선달", 20);
04       Entry<String, String> e2 = new Entry<>("기타", "등등");
05
06       // Entry<int, String> e3 = new Entry<>(30, "아무개");
07
08       System.out.println(e1.getKey() + " " + e1.getValue());
09       System.out.println(e2.getKey() + " " + e2.getValue());
10    }
11  }
```

<Integer>와 동일하다.

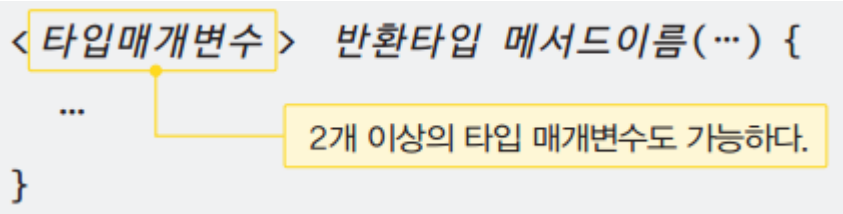
<String>과 동일하다.

타입 매개변수로 기초 타입을 사용할 수 없다.

```
김선달 20
기타 등등
```

제네릭 메서드의 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



- 제네릭 메서드를 호출할 때는 구체적인 타입 생략 가능

- [예제 10-17]
  - 배열의 타입에 상관없이 모든 원소 출력

```
3  public class GenMethod1Demo {
4      static class Utils {
5          public static <T> void showArray(T[] a) {
6              for (T t : a)
7                  System.out.printf("%s ", t);
8              System.out.println();
9          }
10
11         public static <T> T getLast(T[] a) {
12             return a[a.length - 1];
13         }
14     }
15
16     public static void main(String[] args) {
17         Integer[] ia = { 1, 2, 3, 4, 5 };
18         Character[] ca = { 'H', 'E', 'L', 'L', 'O' };
19
20         Utils.showArray(ia);
21         Utils.<Character>showArray(ca);
22
23         System.out.println(Utils.getLast(ia));
24     }
25 }
```

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