Chapter 13. 제네릭

13.1 제네릭이란?

제네릭(Generic) : 결정되지 않은 타입을 파라미터로 처리하고 실제 사용할 때 파라미터를 구체적인 타입으로 대체시키는 기능

Box.java

```
package ch13.sec01;

// 타입 파라미터로 T 사용
public class Box<T> {
    public T content;
}
```

GenericExample.java

```
package ch13.sec01;

public class GenericExample {
    public static void main(String[] args) {

        // 타입 파라미터 T 대신 String으로 대체
        //Box<String> box1 = new Box<String>();
        Box<String> box1 = new Box<>();
        box1.content = "안녕하세요.";
        String str = box1.content;
        System.out.println(str);

        // 타입 파라미터 T 대신 Integer로 대체
        //Box<Integer> box2 = new Box<Integer>();
        Box<Integer> box2 = new Box<>();
        box2.content = 100;
        int value = box2.content;
        System.out.println(value);
    }
}
```

- 기본 타입은 타입 파라미터의 대체 타입이 될 수 없음

13.2 제네릭 타입

- **제네릭 타입** : 결정되지 않은 타입을 파라미터로 가지는 클래스와 인터페이스.
- 타입 파라미터는 일반적으로 대문자 알파벳 한 글자로 표현.
- 외부에서 제네릭 타입을 사용하려면 타입 파라미터에 구체적인 타입 지정해야 함.
- 만약 지정하지 않을 시 `Object` 타입 암묵적으로 사용.

Product.java

```
package ch13.sec02.exam01;

//제네릭 타입 (타입 파라미터로 K, M 정의)
public class Product<K, M> {
    //필드
    private K kind;
    private M model;

    //메소드 (타입 파라미터를 리턴 타입과 매개 변수 타입으로 사용)
    public K getKind() { return this.kind; }
    public M getModel() { return this.model; }
    public void setKind(K kind) { this.kind = kind; }
    public void setModel(M model) { this.model = model; }
}
```

- `Product`에 다양한 종류와 모델 제품 저장하기 위해 타입 파라미터 사용.

Tv.java

```
package ch13.sec02.exam01;

public class Tv {
}
```

Car.java

```
package ch13.sec02.exam01;

public class Car {
}
```

GenericExample.java

```
package ch13.sec02.exam01;

public class GenericExample {
    public static void main(String[] args) {
        //K는 Tv로 대체, M은 String으로 대체
        Product<Tv, String> product1 = new Product<>();

        //Setter 매개값은 반드시 Tv와 String을 제공
        product1.setKind(new Tv());
        product1.setModel("스마트Tv");
```

```
        //Getter 리턴값은 Tv와 String이 됨
        Tv tv = product1.getKind();
        String tvModel = product1.getModel();

        //------------------------------------------------------------------------

        //K는 Car로 대체, M은 String으로 대체
        Product<Car, String> product2 = new Product<>();

        //Setter 매개값은 반드시 Car와 String을 제공
        product2.setKind(new Car());
        product2.setModel("SUV자동차");

        //Getter 리턴값은 Car와 String이 됨
        Car car = product2.getKind();
        String carModel = product2.getModel();
    }
}
```

13.3 제네릭 메소드

- 타입 파라미터를 가지고 있는 메소드
- 매개값이 어떤 타입이냐에 따라 컴파일 과정에서 구체적인 타입으로 대체

Box.java

```
package ch13.sec03.exam01;

public class Box<T> {
    //필드
    private T t;

    //Getter 메소드
    public T get() {
        return t;
    }

    //Setter 메소드
    public void set(T t) {
        this.t = t;
    }
}
```

GenericExample.java

```
package ch13.sec03.exam01;

public class GenericExample {
```

```
    //제네릭 메소드
    public static <T> Box<T> boxing(T t) {
        Box<T> box = new Box<T>();
        box.set(t);
        return box;
    }

    public static void main(String[] args) {
        //제네릭 메소드 호출
        Box<Integer> box1 = boxing(100);
        int intValue = box1.get();
        System.out.println(intValue);

        //제네릭 메소드 호출
        Box<String> box2 = boxing("홍길동");
        String strValue = box2.get();
        System.out.println(strValue);
    }
}
```

13.4 제한된 타입 파라미터 (bounded type parameter)

- 모든 타입으로 대체할 수 없고, 특정 타입과 자식 또는 구현 관계에 있는 타입만 대체할 수 있는 타입 파라미터

```
// Number 타입과 자식 클래스 (Byte, Short, Integer, Long, Double)에만 대체 가능한 타입 파라미터
public <T extends Number> boolean compare(T t1, T t2) {
    double v1 = t1.doubleValue();
    double v2 = t2.doubleValue();
    // Object의 메소드뿐만 아니라 Number가 가지고 있는 메소드도 사용 가능

    return (v1 == v2);
}
```

GenericExample.java

```
package ch13.sec04;

public class GenericExample {
    //제한된 타입 파라미터를 갖는 제네릭 메소드
    public static <T extends Number> boolean compare(T t1, T t2) {
        //T의 타입을 출력
        System.out.println("compare(" + t1.getClass().getSimpleName() + ", " +
                t2.getClass().getSimpleName() + ")");

        //Number의 메소드 사용
        double v1 = t1.doubleValue();
```

```
        double v2 = t2.doubleValue();

        return (v1 == v2);
    }

    public static void main(String[] args) {
        //제네릭 메소드 호출
        boolean result1 = compare(10, 20);
        System.out.println(result1);
        System.out.println();

        //제네릭 메소드 호출
        boolean result2 = compare(4.5, 4.5);
        System.out.println(result2);
    }
}
```

결과

```
compare(Integer, Integer)
false

compare(Double, Double)
true
```

13.5 와일드카드 타입 파라미터

- 제네릭 타입을 매개값이나 리턴 타입으로 사용할 때 타입 파라미터로 ?(와일드카드) 사용 가능
- ?는 범위에 있는 모든 타입으로 대체할 수 있다는 표시

Person.java

```
package ch13.sec05;

public class Person {
}

class Worker extends Person {
}

class Student extends Person {
}

class HighStudent extends Student {
}

class MiddleStudent extends Student{
}
```

Applicant.java

```
package ch13.sec05;

public class Applicant<T> {
    public T kind;

    public Applicant(T kind) {
        this.kind = kind;
    }
}
```

Course.java

```
package ch13.sec05;

public class Course {
    //모든 사람이면 등록 가능
    public static void registerCourse1(Applicant<?> applicant) {
        System.out.println(applicant.kind.getClass().getSimpleName() +
                "이(가) Course1을 등록함");
    }

    //학생만 등록 가능
    public static void registerCourse2(Applicant<? extends Student> applicant) {
        System.out.println(applicant.kind.getClass().getSimpleName() +
                "이(가) Course2를 등록함");
    }

    //직장인 및 일반인만 등록 가능
    public static void registerCourse3(Applicant<? super Worker> applicant) {
        System.out.println(applicant.kind.getClass().getSimpleName() +
                "이(가) Course3을 등록함");
    }
}
```

GenericExample.java

```
package ch13.sec05;

public class GenericExample {
    public static void main(String[] args) {
        //모든 사람이 신청 가능
        Course.registerCourse1(new Applicant<Person>(new Person()));
        Course.registerCourse1(new Applicant<Worker>(new Worker()));
        Course.registerCourse1(new Applicant<Student>(new Student()));
        Course.registerCourse1(new Applicant<HighStudent>(new HighStudent()));
```

```
        Course.registerCourse1(new Applicant<MiddleStudent>(new MiddleStudent()));
        System.out.println();

        //학생만 신청 가능
        //Course.registerCourse2(new Applicant<Person>(new Person())); (x)
        //Course.registerCourse2(new Applicant<Worker>(new Worker())); (x)
        Course.registerCourse2(new Applicant<Student>(new Student()));
        Course.registerCourse2(new Applicant<HighStudent>(new HighStudent()));
        Course.registerCourse2(new Applicant<MiddleStudent>(new MiddleStudent()));
        System.out.println();

        //직장인 및 일반인만 신청 가능
        Course.registerCourse3(new Applicant<Person>(new Person()));
        Course.registerCourse3(new Applicant<Worker>(new Worker()));
        //Course.registerCourse3(new Applicant<Student>(new Student()));
(x)
        //Course.registerCourse3(new Applicant<HighStudent>(new HighStudent()));
(x)
        //Course.registerCourse3(new Applicant<MiddleStudent>(new
MiddleStudent()));  (x)
    }
}
```