제네릭

컴퓨터공학전공
박요한

수업내용

- 제네릭의 이해
- 제네릭의 기본 문법

제네릭의 이해

- 제네릭이란?
  - ✓사전적 의미 : 포괄적인, 회사 이름이 붙지 않은, 일반 명칭으로 판매되는

- 제네릭 in OOP?
  - ✓타입을 매개변수(파라미터)화 해서 컴파일시 구체적인 타입이 결정되도록 하는 것
  - ✓즉, 여러 자료형이 대체되도록 프로그래밍 하는 것
  - ✓변수가 하나의 자료형에 국한되지 않고 여러 자료형에 쓰일 수 있도록 프로그래밍 하는 방식

AppleBox와 OrangeBo

```
class Apple {
    public String toString() {
        return "I am an apple.";
    }
}

class Orange {
    public String toString() {
        return "I am an orange.";
    }
}
```

```
class AppleBox {
    private Apple ap;

    public void set(Apple a) {
        ap = a;
    }

    public Apple get() {
        return ap;
    }
}


class OrangeBox {
    private Orange or;

    public void set(Orange o) {
        or = o;
    }

    public Orange get() {
        return or;
    }
}
```

```
class AppleBox {
    private Apple ap;

    public void set(Apple a) {
        ap = a;
    }

    public Apple get() {
        return ap;
    }
}


class OrangeBox {
    private Orange or;

    public void set(Orange o) {
        or = o;
    }

    public Orange get() {
        return or;
    }
}
```

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

        // 과일 담는 박스 생성
        AppleBox aBox = new AppleBox();
        OrangeBox oBox = new OrangeBox();

        // 과일을 박스에 담는다.
        aBox.set(new Apple());
        oBox.set(new Orange());

        // 박스에서 과일을 꺼낸다.
        Apple ap = aBox.get();
        Orange og = oBox.get();

        System.out.println(ap);
        System.out.println(og);
    }
}
```

```
class Box {
    private Object ob;

    public void set(Object o) {
        ob = o;
    }
    public Object get() {
        return ob;
    }
}
```

```
class FruitAndBox2 {
    public static void main(String[] args) {
        Box aBox = new Box();
        Box oBox = new Box();

        // 과일을 박스에 담는다.
        aBox.set(new Apple());
        oBox.set(new Orange());

        // 박스에서 과일을 꺼낸다.
        Apple ap = (Apple)aBox.get();
        Orange og = (Orange)oBox.get();

        System.out.println(ap);
        System.out.println(og);
    }
}
```

```
class Box {
    private Object ob;

    public void set(Object o) {
        ob = o;
    }
    public Object get() {
        return ob;
    }
}
```

```
class FruitAndBoxFault {
    public static void main(String[] args) {
        Box aBox = new Box();
        Box oBox = new Box();

        // 과일을 박스에 담은 것일까?
        aBox.set("Apple");
        oBox.set("Orange");

        // 박스에서 과일을 제대로 꺼낼 수 있을까?
        Apple ap = (Apple)aBox.get();
        Orange og = (Orange)oBox.get();

        System.out.println(ap);
        System.out.println(og);
    }
}
```

```
class Box {
    private Object ob;

    public void set(Object o) {
        ob = o;
    }
    public Object get() {
        return ob;
    }
}
```

```
class FruitAndBoxFault2 {
    public static void main(String[] args) {
        Box aBox = new Box();
        Box oBox = new Box();

        // 과일을 박스에 담은 것일까?
        aBox.set("Apple");
        oBox.set("Orange");

        System.out.println(aBox.get());
        System.out.println(oBox.get());
    }
}
```

```
class Box {
    private Object ob;

    public void set(Object o) {
        ob = o;
    }
    public Object get() {
        return ob;
    }
}
```

```
class Box<T> {
    private T ob;

    public void set(T o) {
        ob = o;
    }
    public T get() {
        return ob;
    }
}
```

```
class FruitAndBoxFault_Generic {
    public static void main(String[] args) {
        Box<Apple> aBox = new Box<Apple>();
        Box<Orange> oBox = new Box<Orange>();

        // 과일을 박스에 담은 것일까?
        aBox.set("Apple");
        oBox.set("Orange");

        // 박스에서 과일을 제대로 꺼낼 수 있을까?
        Apple ap = aBox.get();
        Orange og = oBox.get();

        System.out.println(ap);
        System.out.println(og);
    }
}
```

```java
class Box<T> {
    private T ob;

    public void set(T o) {
        ob = o;
    }
    public T get() {
        return ob;
    }
}
```

```java
class FruitAndBox2_Generic {
    public static void main(String[] args) {
        Box<Apple> aBox = new Box<Apple>();
        Box<Orange> oBox = new Box<Orange>();

        // 과일을 박스에 담는다.
        aBox.set(new Apple());
        oBox.set(new Orange());

        // 박스에서 과일을 꺼내는데 형 변환 하지 않는다.
        Apple ap = aBox.get();
        Orange og = oBox.get();

        System.out.println(ap);
        System.out.println(og);
    }
}
```

1. 왜 제네릭을 사용해야 하는가?

- 제네릭(Generic) 타입이란?
  - ✓ '컴파일 단계'에서 '잘못된 타입 사용될 수 있는 문제'제거 가능
  - ✓ 자바5부터 새로 추가 !
  - ✓ 컬렉션, 람다식(함수적 인터페이스), 스트림, NIO에서 널리 사용
  - ✓ 제네릭을 모르면 API 도큐먼트 해석 어려우므로 학습 필요

Class ArrayList<E>

```
default BiConsumer<T,U> andThen(BiConsumer<? super T,? super U> after)
```

1. 왜 제네릭을 사용해야 하는가?

- 제네릭을 사용하는 코드의 이점
  - ✓컴파일 시 강한 타입 체크 가능
    - 실행 시 타입 에러가 나는 것 방지
    - 컴파일 시에 미리 타입을 강하게 체크해서 에러 사전 방지
  - ✓타입변환 제거 가능

```
List list = new ArrayList();
list.add("hello");
String str = (String) list.get(0);
```

```
List<String> list = new ArrayList<String>();
list.add("hello");
String str = list.get(0);
```

2. 제네릭 타입

- 제네릭 타입이란?
  - ✓타입을 파라미터로 가지는 클래스와 인터페이스
  - ✓선언 시 클래스 또는 인터페이스 이름 뒤에 “<>” 부호 붙임
  - ✓“<>” 사이에는 타입 파라미터 위치

  - ✓타입 파라미터
    - 일반적으로 대문자 알파벳 한 문자로 표현
    - 개발 코드에서는 타입 파라미터 자리에 구체적인 타입을 지정해야

2. 제네릭 타입

- 제네릭 타입 사용 여부에 따른 비교
  - ✓제네릭 타입을 사용하지 않은 경우
    - Object 타입 사용 → 빈번한 타입 변환 발생 → 프로그램 성능 저하

```
public class Box {
   private Object object;
   public void set(Object object) { this.object = object; }
   public Object get() { return object; }
}
```

```
Box box = new Box();
box.set("hello");                  //String 타입을 Object 타입으로 자동 타입 변환해서 저장
String str = (String) box.get();   //Object 타입을 String 타입으로 강제 타입 변환해서 얻음
```

2. 제네릭 타입

- 제네릭 타입 사용 여부에 따른 비교
  - ✓제네릭 타입 사용한 경우
    - 클래스 선언할 때 타입 파라미터 사용
    - 컴파일 시 타입 파라미터가 구체적인 클래스로 변경

```
public class Box<T> {
    private T t;
    public T get() { return t; }
    public void set(T t) { this.t = t; }
}
```

```
Box<String> box = new Box<String>();
```

```
public class Box<String> {
    private String t;
    public void set(String t) { this.t = t; }
    public String get() { return t; }
}
```

```
Box<String> box = new Box<String>();
box.set("hello");
String str = box.get();
```

```
Box<Integer> box = new Box<Integer>();
```

```
public class Box<Integer> {
    private Integer t;
    public void set(Integer t) { this.t = t; }
    public Integer get() { return t; }
}
```

```
Box<Integer> box = new Box<Integer>();
box.set(6);
int value = box.get();
```

3. 멀티 타입 파라미터

- 제네릭 타입은 두 개 이상의 타입 파라미터 사용 가능
  - ✓각 타입 파라미터는 콤마로 구분
    - Ex) class<K, V, ···> { ··· }
    - interface<K, V, ···> { ··· }

```
public class Product<T, M> {
    private T kind;
    private M model;

    public T getKind() { return this.kind; }
    public M getModel() { return this.model; }

    public void setKind(T kind) { this.kind = kind; }
    public void setModel(M model) { this.model = model; }
}
```

```
Product<Tv, String> product = new Product<Tv, String>();
```

  - ✓자바 7부터는 다이아몬드 연산자 사용해 간단히 작성과 사용 가능

```
Product<Tv, String> product = new Product<>();
```

4. 제네릭 메소드

- 제네릭 메소드
  - ✓매개변수 타입과 리턴 타입으로 타입 파라미터를 갖는 메소드
  - ✓제네릭 메소드 선언 방법
    - 리턴 타입 앞에 “<>” 기호를 추가하고 타입 파라미터 기술
    - 타입 파라미터를 리턴 타입과 매개변수에 사용

```
public <타입파라미터,...> 리턴타입 메소드명(매개변수,...) { ... }
```

```
public <T> Box<T> boxing(T t) { ... }
```

  - ✓제네릭 메소드 호출하는 두 가지 방법

```
리턴타입 변수 = <구체적타입> 메소드명(매개값);      //명시적으로 구체적 타입 지정
리턴타입 변수 = 메소드명(매개값);                   //매개값을 보고 구체적 타입을 추정
```

```
Box<Integer> box = <Integer>boxing(100);      //타입 파라미터를 명시적으로 Integer 로 지정
Box<Integer> box = boxing(100);               //타입 파라미터를 Integer 으로 추정
```

4. 제네릭 메소드

```
class BoxFactory {
    public static <T> Box<T> makeBox(T o) {
        Box<T> box = new Box<T>();
        box.set(o);
        return box;
    }
}
```

```
class GenericMethodBoxMaker {
    public static void main(String[] args) {
        Box<String> sBox = BoxFactory.makeBox("Sweet");
        System.out.println(sBox.get());

        Box<Double> dBox = BoxFactory.makeBox(7.59);
        System.out.println(dBox.get());
    }
}
```